МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

О.С. Зеленський
В.С. Лисенко

РОЗРОБКА WINDOWS-ДОДАТКІВ НА МОВІ C#

Частина 2
Навчальний посібник

Кривий Ріг
2023

Цей посібник є другою частиною вивчення мови C# для розробки Windos-додатків. У першій частині [2] детально розглядається реалізація графічного виведення на дисплей і принтер, керування клавіатурою, мишкою і таймером. Закладаються основи розробки програм із застосуванням Windows Forms.

У даному посібнику (другій частині) будуть детально вивчатися елементи управління, розглянуті робота з базою даних, Web-програмування, використання технології WPF.

Оскільки мова C# має широкі можливості доцільно назву посібника строго не "прив'язувати" до назви конкретної дисципліни і продовжувати випуск наступних частин посібника. При цьому кожна частина посібника повинна бути функціонально незалежною, що дозволяє її використовувати для конкретної дисципліни.

При викладанні матеріалу використані фрагменти програм книг: Чарльз Петцольд " Программирование для Microsoft Windows на C#" том 2 [3], Эндрю Стиллмен и Дженнифер "Изучаем C#" [4], окремі матеріали із Internet. Багато прикладів скореговані з врахуванням сучасної .NET FrameWork 6. Дається значна кількість авторських розробок.

Навчальний посібник адресовано студентам, аспірантам, викладачам. Може бути використаний як самовчитель.

Автори: Зеленський О.С., Лисенко В.С. – Кривий Ріг: Державний університет економіки і технологій, 2023.-357 с.

Рецензенти:

А.І. Купін, д.т.н, професор, завідувач кафедри комп'ютерних систем та мереж, Криворізький національний університет.

І.О. Музика, к.т.н, доцент кафедри комп'ютерних систем та мереж, декан факультету інформаційних технологій Криворізький національний університет.

В.Б. Хоцкіна, к.т.н., доцент кафедри інформатики і прикладного програмного забезпечення, Державний університет економіки і технологій.

Рекомендовано Вченою радою Державного університету економіки і технологій

Протокол № 10 від 30.03.2023 р.

Зміст

Вступ

За останні декілька років *Web*-орієнтовані додатки стали надзвичайно популярними. Можливість розміщувати всю логіку додатків на централізованому сервері виглядає дуже привабливою з погляду адміністратора. Розгортання програмного забезпечення, що базується на клієнті, дуже важко, особливо якщо воно засноване на *COM*-об'єктах. Web-орієнтовані програми не повною мірою можуть надати широке коло вимог користувача. Платформа *.NET Framework* дозволяє розробляти як *Web*-програми так і інтелектуальні клієнтські програми з широкими можливостями. У багатьох випадках на стадії проектування виникають проблеми прийняття рішення розробки *Web*-орієнтованого або клієнтського додатка. Вибір залежить від призначення програми, що розробляється.

Не варто піддаватися помилкам через наявність у назві *Framework* слова "*Net*" і думати, що це середовище призначене лише для створення програм, орієнтованих на Інтернет. Тут слово "*Net*" пояснюється, на думку *Microsoft*, як розробка розподілених додатків, у яких обробка розподіляється між клієнтом та сервером. Це є значним кроком уперед. У *Framework* можна розробляти ефективні клієнтські та розподілені програми.

У посібнику розглядається розробка клієнтських *Windows*-додатків. Вони можуть бути розроблені швидко та ефективно. У цьому дається докладний опис типових завдань, які зустрічаються у практиці програмування.

Windows Forms знайома студентам, оскільки є досвід розробки *Visual Basic* і *Visual C++*. Тут створюються нові форми (вікна чи діалоги), розміщуються елементи керування з панелі інструментів на поверхню візуального дизайнера форм (*Form Designer*). У роботі розглянуто такі аспекти *Windows Forms*:

1. Клас *Form*.
2. Ієрархія класів *Windows Forms*.
3. Елементи управління і компоненти, які є частиною простору імен *System.Windows.Forms*.
4. Меню і панелі інструментів.
5. Створення елементів управління.

Детально розглядається робота з базою даних з використанням технології *ADO .NET*, створення клієнт-серверних програм на основі технології *ASP .NET*. Даються основи мови *XAML*, використання бібліотеки *WPF* для створення інтерфейсів для інтелектуальних клієнтських додатків.

1. Створення додатку Windows Forms

Перше, що необхідно зробити – створити додаток *Windows Forms*. Наприклад створимо порожню форму і відобразимо її на екрані. Під час розробки цього прикладу ми використовуватимемо *Visual Studio .NET*. Наберемо його в текстовому редакторі та зберемо за допомогою компілятора командного рядка. Нижче наведено код прикладу.

```
using System;
using System.Windows.Forms;

namespace NotepadForms
 {
  public class MyForm : System.Windows.Forms.Form
  {
            public MyForm()
      { }
          [STAThread]
  static void Main()
  {
            Application.Run(new MyForm());
  }
}
```

Коли ми скомпілюємо і запустимо цей приклад, отримаємо маленьку порожню форму без заголовка. Жодних реальних функцій, але це – *Windows Forms*.

У наведеному коді заслуговують на увагу дві речі. Перша – те що, що при створенні класу *MyForm* використовується спадкування. Наступний рядок оголошує *MyForm* як спадкоємця *System.Windows.Forms.Form*:

```
public class MyForm : System.Windows.Forms.Form
```

Клас *Form* – один з головних класів у просторі імен *System.Windows.Forms*. Наступний фрагмент коду варто розглянути докладніше:

```
[STAThread]
 static void Main()
 {
    Application.Run(new MyForm());
 }
```

Main – точка входу за замовчуванням у будь-яку клієнтську програму на *C#*. Як правило, у великих додатках метод *Main()* не перебуватиме в класі форми, а скоріше в класі, що відповідає за процес запуску. У цьому випадку ви повинні встановити ім’я такого класу, що запускає додаток, у діалоговому вікні властивостей проекту. Зверніть увагу на атрибут [*STAThread*]. Він встановлює модель багатопоточності *COM* у *STA* (однопоточний

апартамент). Модель багатопоточності *STA* потрібна для взаємодії з *COM* і встановлюється за замовчуванням у кожному проекті *Windows Forms*.

Метод *Application.Run()* відповідає за запуск стандартного циклу повідомлень додатку. *Application.Run()* та має три перевантаження.

Перше з них не приймає параметрів; друге приймає як параметр об'єкт *ApplicationContext*. У нашому прикладі об'єкт *MyForm* стає головною формою програми. Це означає, що коли форма закривається, програма завершується. Використовуючи клас *ApplicationContext*, можна контролювати завершення головного циклу повідомлень і вихід з програми.

Клас *Application* містить дуже корисну функціональність. Він надає групу статичних методів і властивостей для керування процесом запуску та зупинки програми, а також забезпечує доступ до повідомлень *Windows*, що обробляється програмою.

У табл. 1.1. перелічені деякі з цих найбільш корисних методів і властивостей.

Таблиця 1.1

Деякі корисні методи і властивості класу *Application*

Метод/властивість	**Опис**
CommonAppDataPath	Шлях до даних, загальний для всіх користувачів додатку. Зазвичай *це БазовийШлях\Назва компанії\Назва продукту\Версія*, де *БазовийШлях - C:\Documents and Settings\ім'я користувача\ ApplicationData.* Якщо шлях не існує, він буде створений.
ExecutablePath	Шлях та ім'я виконуваного файлу, що запускає додаток.
LocalUserAppDataPath	Подібно *CommonAppDataPath*, але з тією відмінністю, що підтримується *роуминг* (переміщення).
MessageLoop	*True* або *false* – в залежності від того, чи існує цикл повідомлень у поточному потоці.
StartupPath	Подібно *ExecutablePath*, з тією відмінністю, що ім'я файлу не повертається.
AddMessageFilter	Використовується для попередньої обробки повідомлень. Об'єкт, що реалізує *IMessageFilter*, дозволяє фільтрувати повідомлення в циклі або організувати спеціальну обробку, що виконується перед тим, як повідомлення потрапить до циклу.
DoEvents	Дозволяє обробити повідомлення у черзі.
Exit і ExitThread	*Exit* завершує поточний цикл повідомлень і викликає вихід із програми. *ExitThread* завершує цикл повідомлень та закриває всі вікна поточного потоку.

А тепер як виглядатиме ця програма, якщо її згенерувати у *Visual Studio* 2019. Перше, що слід зазначити – буде створено два файли. Причина в тому, що *Visual Studio* використовує можливість часткових класів і виділяє весь код, згенерований візуальним дизайнером, в окремий файл. Якщо використовується ім'я за замовчуванням – *Form1*, то ці два файли будуть називатися *Form1.cs* та *Form1.Designer.cs*. Нижче наведено код цих двох файлів, згенерованих *Visual Studio*. Спочатку – *Form1.cs*:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data; using System.Drawing;
using System.Linq; using System.Text;
using System.Windows.Forms;

namespace VisualStudioForm
{
    public partial class Form1 : Form
      {       public Form1()

          {
             InitializeComponent();
          }
      }
}
```

Тут ми бачимо лише оператори *using* і простий конструктор. А ось код *Form1. Designer.cs:*

```
namespace VisualStudioForm
{
   partial class Form1
   {

 /// <summary>
 /// Required designer variable.
 /// </summary>
private System.ComponentModel.IContainer components = null;
/// <summary>
/// Clean up any resources being used.
/// </summary>
/// < param name="disposing" > true if
managed resources should be disposed;
otherwise, false. < /param >

protected override void Dispose(bool disposing)
 {
   if (disposing && (components != null))
   {
       components.Dispose();
    }
base.Dispose(disposing);
 }

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
```

```
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.Text = "Form1";
        }
        #endregion
    }
}
```

Файл, згенерований дизайнером форм, рідко піддається ручному редагуванню. Єдиним винятком може бути випадок, коли потрібна спеціальна обробка методом *Dispose()*. Метод *InitializeComponent* ми обговоримо пізніше. Якщо поглянути на цей приклад коду програми в цілому, ми побачимо, що він набагато довший, ніж простий приклад командного рядка. Тут перед початком класу кілька операторів *using*, більшість з них в даному прикладі не потрібна. Їх присутність не заважає. Клас Form1 успадковується від System.Windows.Forms.Form.

У файлі *Form1.Designer* наводиться рядок:

```
private System.ComponentModel.IContainer components = null;
```

У даному прикладі цей рядок коду нічого не робить. Але, додаючи компонент до форми, ви можете також додати його до об’єкту *components*, який являє собою контейнер. Причина додавання цього контейнера - в необхідності правильної обробки знищення форми. Клас форми підтримує інтерфейс *IDisposable,* тому що його реалізовано в класі *Component*. Коли компонент додається до контейнеру, то цей контейнер повинен реалізувати коректне знищення свого вмісту при закритті форми. Це можна побачити в методі Dispose, який наведено в нашому прикладі:

```
protected override void Dispose(bool disposing)
 {
    if (disposing && (components != null))
       {
         components.Dispose();
       }
    base.Dispose(disposing);
}
```

Тут ми бачимо, що коли викликається метод форми Dispose, то також викликається метод Dispose об’єкта components, оскільки він містить інші компоненти, які також повинні бути коректно видалені. Конструктор класу Form1, який знаходиться у файлі Form1.cs, виглядає так:

```
public Form1()
  {
    InitializeComponent();
  }
```

Зверніть увагу на виклик *InitializeComponent()*. Метод *InitializeComponent()* знаходиться у файлі *Form1.Designer.cs* і робить те, що випливає з його найменування - ініціалізує всі елементи керування, які можна додати до форми. Він також ініціалізує властивості форми. У нашому прикладі метод *InitializeComponent()* виглядає так:

```
private void InitializeComponent()
 {
   this.components = new System.ComponentModel.Container();
   this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
   this.Text = "Form1";
 }
```

Як бачите, тут є лише базовий код ініціалізації. Цей метод пов'язаний із візуальним дизайнером *Visual Studio*. Коли до форми вносяться зміни в дизайнері, вони дублюються в *InitializeComponent()*. Якщо ви вносите будь-які зміни в *InitializeComponent()*, то наступного разу після того, як щось буде змінено в дизайнері, ці ручні зміни будуть втрачені. *InitializeComponent()* повторно генерується після кожної зміни дизайну форми. Якщо виникає необхідність додати деякий додатковий код для форми або елементів керування та компонентів форми, це має бути зроблено після виклику *InitializeComponent()*. Цей метод також відповідає за створення екземплярів елементів керування, тому будь-який виклик, що посилається на них, виконаний до *InitializeComponent()*, завершиться викликом винятку нульового посилання.

Щоб додати елемент керування або компонент до форми, натисніть комбінацію клавіш *<Ctrl+Alt+X>* або виберіть пункт меню *View.Toolbox* (Вид->Панель інструментів) у середовищі *Visual Studio .NET*. Натисніть правою кнопкою миші на *Form1.cs* у провіднику Solution Explorer і в контекстному меню виберіть *пункт View Designer* (Показати дизайнер). Виберіть елемент керування *Button* і перетягніть на поверхню форми візуального дизайнера. Можна також двічі натиснути на вибраний елемент керування, і його буде додано до форми. Те саме слід зробити з елементом *TextBox*.

Тепер, після додавання цих двох елементів керування до форми, метод *InitializeComponent()* розширюється і містить наступний код:

```
private void InitializeComponent()
{
this.button1  = new System.Windows.Forms.Button();
this.textBox1 = new System.Windows.Forms.TextBox();
this.SuspendLayout();
 //
 // button1
 //
this.button1.Location = new System.Drawing.Point(77, 137);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(75, 23); this.button1.TabIndex = 0;
```

```
this.button1.Text = "button1";
this.button1.UseVisualStyleBackColor = true;
//
// textBox1
//
this.textBox1.Location = new System.Drawing.Point(67, 75);
this.textBox1.Name = "textBox1";
this.textBox1.Size = new System.Drawing.Size(100, 20);
this.textBox1.TabIndex = 1;
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(284, 264);
this.Controls.Add(this.textBox1);
this.Controls.Add(this.button1);
this.Name = "Form1";
this.Text = "Form1";
this.ResumeLayout(false);
this.PerformLayout();
 }
```

Якщо подивитися на перші три рядки коду цього методу, ми побачимо створення екземплярів елементів керування *Button* і *TextBox*. Зверніть увагу на імена, присвоєні їм - *textBox1* та *button1*. За замовчуванням дизайнер іменує ім'я класу елемента керування, доповнене цілим числом. Коли ви додаєте наступну кнопку, дизайнер називає її *button2* і т.д. Наступний рядок – частина пари *SuspendLayout/ResumeLayout*. Метод *SuspendLayout()* тимчасово припиняє події розміщення, які мають місце при початковій ініціалізації елемента керування. Наприкінці методу викликається *ResumeLayout()*, щоб повернути все до норми. У складній формі з безліччю елементів керування метод *InitializeComponent()* може стати досить великим.

Щоб змінити значення властивостей елемента керування, потрібно або натиснути *<F4>*, або вибрати пункт меню *View->Properties Window* (Вид->Вікно властивостей). Це вікно дозволяє модифікувати більшість властивостей елемента керування чи компонента. При внесенні змін до вікна властивостей метод *InitializeComponent()* автоматично переписується для того, щоб відобразити нові значення властивостей. Наприклад, змінивши властивість *Text* у *MyButton* у вікні властивостей, отримаємо наступний код *InitializeComponent()*:

```
 // button1
//
this.button1.Location = new System.Drawing.Point(77, 137);
this.button1.Name = "button1"; this.button1.Size = new System.Drawing.Size(75, 23);
this.button1.TabIndex = 0;
this.button1.Text = "My Button";
this.button1.UseVisualStyleBackColor = true;
```

Навіть якщо ви надаєте перевагу використовувати який-небудь редактор, відмінний від *Visual Studio .NET*, то напевно захочете вімкнути функції на кшталт *InitializeComponent()* у свої проекти. Збереження всього коду ініціалізації в одному місці забезпечить можливість його виклику в кожному з конструкторів.

2. Клас Control

Важливість розуміння ієрархії класів стає очевидною в процесі проєктування та конструювання власних елементів керування. Якщо такий елемент керування успадкований від конкретного бібліотечного елемента керування, наприклад, коли створюється текстове поле з деякими додатковими методами та властивостями, є сенс успадкувати його від звичайного текстового поля і потім перевизначити і додавати необхідні методи. Однак якщо доводиться створювати елемент керування, який не відповідає жодному з існуючих у *.NET Framewo*rk, то його доведеться успадкувати від одного з базових класів: *Control* або *ScrollableControl*, якщо потрібно використовувати можливості прокручування, або ContainerControl, якщо він повинен бути контейнером для інших елементів керування. Решта цієї глави присвячена вивченню більшості з цих класів - як вони працюють разом, і як їх можна використовувати для побудови клієнтських додатків, що виглядають професійно.

Простір імен *System.Windows.Forms* включає один клас, який є базовим майже для всіх елементів керування та форм - *System.Windows.Forms*. *Control*. Він реалізує основну функціональність створення екранів, які бачить користувач. Клас *Control* успадкований від *System.ComponentModel. Component*. Клас *Component* забезпечує класу *Control* інфраструктуру, необхідну для того, щоб його можна було перетягувати і розмістити на полі дизайнера, а також, щоб він міг включати інші елементи керування. Клас *Control* пропонує величезний обсяг функціональності класам, що успадковуються від нього. Цей список занадто великий, щоб наводити його тут, тому в даному розділі ми розглянемо лише найважливіші можливості, які надає клас *Control*. Пізніше ми розглядатимемо елементи керування, які беруть за основу *Control*, ми побачимо ці методи та властивості в прикладах коду. Наступні підрозділи групують методи та властивості щодо їх функціональності, тому взаємопов’язані елементи можуть бути розглянуті разом.

2.1. Розмір та місце розташування

Розмір і місце розташування елементів керування визначаються властивостями *Height*, *Width*, *Top*, *Bottom*, *Left* і *Right*, разом з *Size* і *Location*, що доповнюють їх. Відмінність полягає в тому, що *Height*, *Width*, *Top*, *Bottom*, *Left* та *Right* приймають одне ціле значення. *Size* набуває значення

структури *Size*, а *Location* - значення структури *Point*. Структури *Size* і *Point* включають координати *X*, *Y*. *Point* зазвичай описує місце розташування, а *Size* - висоту і ширину об’єкта. *Size* та *Point* визначені у просторі імен *System.Drawing*. Обидві структури дуже схожі в тому, що представляють пари координат *X*, *Y*, але, крім того - перевизначені операції, що спрощують порівняння та перетворення. Ви можете, наприклад, складати разом дві структури *Size*. У структурі *Point* операцію додавання перевизначено таким чином, що можна додати до *Point* структуру *Size* і отримати в результаті *Point*. Це дає ефект збільшення відстані до розташування, щоб отримати нове місце, що дуже зручно для динамічного створення форм і елементів керування. Властивість *Bounds* повертає об'єкт *Rectangle*, що представляє екранну область, зайняту елементом керування. Ця область включає смуги прокручування та заголовка. *Rectangle* також відноситься до простору імен *System.Drawing*. Властивість *ClientSize* - структура *Size*, що представляє клієнтську область елемента керування за вирахуванням смуг прокручування та заголовка. Методи *PointToClient* та *PointToScreen* - зручні методи перетворення, які приймають *Point* та повертають *Point*. Метод *PointToClient* приймає структуру Point, що представляє екранні координати, і транслює їх координати поточного клієнтського об'єкта. Це зручно для операцій перетягування. Метод *PointToScreen* виконує зворотну операцію – приймає координати у клієнтському об’єкті та транслює їх у екранні координати. Методи *RectangleToScreen* і *ScreenToRectangle* виконують такі самі операції, але із структурами *Rectangle*, а не *Point*. Властивість *Dock* визначає, до якої межі батьківського елемента керування повинен пристиковуватись цей елемент. Перелік *DockStyle* визначає можливі значення цієї властивості. Вони можуть бути такими: *Top*, *Bottom*, *Right*, *Left*, *Fill* та *None*. Значення *Fill* встановлює розмір даного елемента керування рівним розміру батьківського. Властивість *Anchor* (якір) прикріплює грань даного елемента керування до грані батьківського елемента керування. Це відрізняється від стикування (docking) тим, що не встановлює грань дочірнього елемента керування точно на батьківську грань, а просто витримує постійну відстань між ними. Наприклад, якщо якір правої грані елемента керування встановлений праву грань батьківського елемента, і якщо батько змінює розмір, то права грань даного елемента зберігає постійну дистанцію від правої грані батька - тобто. він змінює розмір разом із батьком. Властивість *Anchor* приймає значення з переліку *AnchorStyle*, а саме: Top, *Bottom*, *Left*, *Right* та *None*. Встановлюючи ці значення, можна змусити елемент керування змінювати свій розмір динамічно разом із батьком. Таким чином, кнопки та текстові поля не будуть усічені або приховані при зміні розмірів форми користувачем. Властивості *Dock* і *Anchor* застосовуються в поєднанні з компонуваннями елементів керування *Flow* і *Table* (вони будуть розглянуті нижче) і дозволяють створювати складні вікна користувача. Зміна розмірів вікна може виявитися непростим завданням для складних форм із безліччю елементів керування. Ці інструменти значно полегшують роботу.

2.2. Зовнішній вигляд

Властивості, що стосуються зовнішнього вигляду елемента керування - це *BackColor* і *ForeColor*, які приймають об'єкт *System.Drawing.Color* як значення. Властивість *BackGroundImage* набуває об'єкта графічного образу як значення. Клас *System.Drawing.Image* - абстрактний клас, який служить базовим для класів *Bitmap* і *Metafile*. Властивість *BackgroundImageLayout* використовує перерахування *ImageLayout* для визначення способу відображення графічного образу елемента керування. Допустимі значення такі: *Center*, *Tile*, *Stretch*, *Zoom* або *None*. Властивості *Font* та *Text* працюють із написами. Щоб змінити *Font*, потрібно створити об'єкт *Font*. Під час створення цього об'єкта вказується ім'я, стиль та розмір шрифту.

2.3. Взаємодія з користувачем

Взаємодія з користувачем найкраще описується серією подій, які генерує елемент керування та на які він реагує. Деякі з найчастіше використовуваних подій: *Click*, *DoubleClick*, *KeyDown*, *KeyPress*, *Validating* та *Paint*. Події, пов'язані з мишею - *Click*, *DoubleClick*, *MouseDown*, *MouseUp*, *MouseEnter*, *MouseLeave* та *MouseHover* - описують взаємодію миші та екранного елемента керування. Якщо ви обробляєте обидві події - *Click* та *DoubleClick* - то щоразу, коли перехоплюється подія *DoubleClick*, також спрацьовує і подія *Click*. Це може призвести до небажаних наслідків при неправильній обробці. До того ж і *Click*, і *DoubleClick* приймають як аргумент *EventArgs*, тоді як події *MouseDown* і *MouseUp* приймають *MouseEventArgs*. Структура *MouseEventArgs* містить кілька частин корисної інформації - наприклад, про кнопку, на якій було виконано натискання, кількість натискань по кнопці, кількість натискань коліщатком миші (за умови його наявності), поточні координати *X* і *Y* курсору миші. Якщо потрібен доступ до будь-якої подібної інформації, то замість подій *Click* або *DoubleClick* потрібно обробляти події *MouseDown* і *MouseUp*. Події клавіатури працюють так. Обсяг необхідної інформації визначає вибір подій, що обробляються. Для найпростіших випадків подія *KeyPress* приймає *KeyPressEventArgs*. Ця структура включає *KeyChar*, що представляє символ натиснутої кнопки. Властивість *Handled* використовується для визначення того, чи подія була оброблена. Встановивши значення *Handled* в *true*, можна домогтися того, що подія не буде передана операційній системі для здійснення стандартної обробки. Якщо потрібна додаткова інформація про клавішу, то більше підійдуть події *KeyDown* або *KeyUp*. Обидва приймають структуру *KeyEventArgs*. Властивості *KeyEventArgs* включають ознаку одночасного стану клавіш <*Ctrl*>, <*Alt*> або <*Shift*>. Властивість *KeyCode* повертає значення типу переліку *Keys*, що ідентифікує клавішу. На відміну від властивості *KeyPressEventArgs.KeyChar*, властивість *KeyCode* повідомляє про кожну клавішу клавіатури, а не лише про буквено-цифрові клавіші. Властивість *KeyData* повертає значення типу *Keys* і встановлює модифікатор.

Значення модифікатора супроводжує значення клавіші, поєднуючись з ним двійковою логічною операцією *"АБО"*. Таким чином, можна отримати інформацію про те, чи одночасно було натиснуто клавішу <*Shift*> або <*Ctrl*>. Властивість *KeyValue* - ціле значення з переліку *Keys*. Властивість *Modifiers* містить значення типу Keys, яке представляє натиснені модифікуючі кнопки. Якщо було натиснуто більше однієї клавіші, їх значення об'єднуються операцією *"АБО"*. Події клавіш надходять у такому порядку:

1. KeyDown
2. KeyPress
3. KeyUp

Події *Validating*, *Validated*, *Enter*, *Leave*, *GotFocus* та *LostFocus* мають відношення до отримання фокусу елементами керування (тобто коли стають активними) та втрати його. Це трапляється, коли користувач натисканням кнопки <*Tab*> переходить до даного елементу керування або вибирає його мишею. Може здатися, що події *Enter*, *Leave*, *GotFocus* та *LostFocus* дуже схожі на те, що вони роблять. Події *GotFocus* та *LostFocus* відносяться до низькорівневих і пов'язані з подіями Windows *WM_SETFOCUS* та *WM_KILLFOCUS*. Зазвичай, коли можливо, краще використовувати події *Enter* і *Leave*. Події *Validating* та *Validated* виникають під час перевірки даних в елементі керування. Ці події приймають аргумент *CancelEventArgs*. З його допомогою можна скасувати наступні події, встановивши властивість *Cancel* у *true*. Якщо ви розробляєте власний код перевірки, і перевірка завершується невдало, то в цьому випадку можна встановити *Cancel* в *true* - тоді елемент керування не втратить фокус. *Validating* відбувається під час перевірки, а *Validated* – після неї. Порядок виникнення подій наступний:

1. Enter
2. GotFocus
3. Leave
4. Validating
5. Validated
6. LostFocus

Розуміння послідовності цих подій є важливим, щоб уникнути рекурсивних ситуацій. Наприклад, спроба встановити фокус елемента керування всередині обробника події LostFocus створює ситуацію взаємоблокування в циклі подій, і програма перестає реагувати на зовнішні дії.

2.4. Функціональність Windows

Простір імен *System.Windows.Forms* - один з небагатьох, що покладається на функціональність операційної системи *Windows*. Клас *Control* – гарний тому приклад. Якщо виконати дизасемблювання *System.Windows.Forms.dll*, можна побачити список посилань на клас *UnsafeNativeMethods*. Середовище *.NET Framework* використовує цей клас як оболонку для всіх стандартних викликів *Win32 API*. Завдяки можливості взаємодії з *Win32 API*, зовнішній вигляд та поведінку стандартної програми *Windows* можна забезпечити засобами простору імен *System.Windows.Forms*. Функціональність, яка підтримує взаємодію з Windows, включає властивості *Handle* та *IsHandleCreated*. Властивість *Handle* повертає *IntPtr*, що містить елемент *HWND* (дескриптор вікна) керування. Дескриптор вікна - це *HWND*, що унікально ідентифікує вікно. Елемент керування може розглядатися як вікно, тому він має відповідний *HWND*. Властивість *Handle* можна використовувати для звернення до будь-яких викликів *Win32 API*. Для отримання доступу до внутрішніх повідомлень Windows можна перевизначити метод *WndProc*. Метод *WndProc* приймає як параметр об'єкт *Message*. Цей об'єкт є просто оболонкою для повідомлення вікна. Він містить властивості *HWnd*, *LParam*, *WParam*, *Msg* та *Result*. Якщо потрібно, щоб повідомлення було оброблено системою, його потрібно буде передати на обробку базовому методу *base.WndProc(msg)*. Якщо потрібно обробити його у вашому додатку спеціальним чином, то передавати його цьому методу недоцільно.

Деякі функції, які дещо складніше класифікувати – це можливості прив'язування даних. Властивість *BindingContext* повертає об'єкт *BindingManagerBase*. Колекція *DataBindings* підтримує *ControlBindingsCollection*, яка представляє колекцію прив'язаних об'єктів елемента керування. *CompanyName*, *ProductName* і *ProductVersions* надають дані про походження елемента керування та його поточної версії. Метод *Invalidate* дозволяє оголосити видиму область елемента керування недійсною, щоб ініціювати її перемальовування. Це можна зробити з цілим елементом керування або його частиною. Після цього, повідомлення перемальовки надсилається методом *WndProc* цього елемента керування. Можна одночасно оголосити недійсним будь-який дочірній елемент керування. Клас *Control* складається з десятків інших властивостей, методів та подій. Наведений список представляє лише деякі з найчастіше використовуваних і призначений для того, щоб дати вам уявлення про доступну функціональність.

3. Форми

Раніше було висвітлено, як створювати просту Windows-програму. Приклад містить один клас, успадкований від *System.Windows.Forms.Form*. Згідно з документацією *.NET Framework*, "форма - це представлення будь-якого вікна у вашому додатку". При вивченні *Visual Basic* термін форма нам знайомий. В області *C++* із застосуванням *MFC*, форми це вікна, діалоги чи фрейми. Можна сміливо сказати, що форма - це основний засіб взаємодії з користувачем. Раніше в цьому розділі вже розкривалися деякі з найчастіше використовуваних властивостей, методів і подій класу *Control*, а оскільки клас *Form* є спадкоємцем *Control*, ті ж методи, властивості та події присутні також у класі *Form*. Клас *Form* додає значний обсяг функціональності до тієї, що забезпечена класом *Control*, і ми поговоримо про неї у цьому розділі.

3.1. Клас Form

Клієнтські *Windows*-додатки можуть містити від однієї до сотень форм. Форми можуть бути засновані на *SDI (Single Document Interface - однодокументний інтерфейс)* або ж на *MDI (Multiple Document Interface - багатодокументний інтерфейс)*. Незалежно від цього, серцем *Windows*-клієнта залишається клас *System.Windows.Forms.Form*.

Клас *Form* успадкований від *ContainerControl*, який, у свою чергу, успадкований від *ScrollableControl* - прямого нащадка *Control*. Звідси можна припустити, що форма може бути контейнером для інших елементів керування, надавати можливість прокручування вмісту, коли він не вміщається в клієнтській області, а також мати безліч тих самих властивостей, методів і подій, які притаманні іншим елементам керування. Все це робить клас *Form* досить складним. У цьому розділі ми розглянемо більшу частину його функціональності.

Важливо добре зрозуміти процес створення форми. Те, що ви хочете зробити залежить від того, де ви напишете ініціалізаційний код. При створенні екземпляра форми події відбуваються у такому порядку:

- конструктор
- Load
- Activated
- Closing
- Closed
- Deactivate

Перші три події стосуються ініціалізації. Те, якому типу ініціалізації ви надаєте перевагу, може визначити подію, в яку має бути поміщений необхідний код. Конструктор класу спрацьовує під час створення екземпляра об’єкта. Подія *Load* відбувається після створення екземпляра, але перед

появою форми на екрані. Різниця між ними пов’язана із існуванням форми. Коли виникає подія *Load*, форма вже існує, хоча поки що й невидима. Під час виконання конструктора форма перебуває у процесі створення. Подія *Activated* відбувається, коли форма стає видимою та поточною. Бувають ситуації, коли цей порядок може бути трохи змінено. Якщо під час конструювання властивості *Visible* встановити значення *true* або викликати метод *Show* (що встановлює *Visible true*), то подія *Load* виникає негайно. А оскільки це робить форму видимою і поточною, також відразу виникає подія *Activated*. Якщо є код після встановлення *Visible* у true, він виконується. Тому послідовність подій виглядатиме приблизно так:

- *конструктор, до виразу Visible = true*
- *Load*
- *Activate*
- *конструктор, після виразу Visible = true*

Потенційно це може спричинити деякі небажані результати. З погляду передового досвіду може здатися, що виконати якнайбільше робіт з ініціалізації в конструкторі - гарна ідея. Але що відбувається, коли форма закривається? Подія *Closing* надає можливість перервати процес. *Closing* приймає параметр *CancelEventArgs*. Цей аргумент має властивість *Cancel*, яка, яка перебуває у стані *true*, перериває подію та залишає форму відкритою. Подія *Closing* виникає при спробі закрити форму, тоді як подія *Closed* - після її закриття. Обидва дають можливість виконати необхідне очищення. Зазначимо, що подія *Deactivate* відбувається після закриття форми. Це ще одне потенційне джерело важких помилок. Завжди потрібно переконатися, що ви не робите нічого такого в *Deactivate*, що завадило б нормальному очищенню форми збирачем сміття. Так, наприклад, встановлення посилання на інший об’єкт може стати причиною того, що форма залишиться в пам’яті. Якщо виконується виклик методу *Application.Exit()*, коли відкрито одну або більше форм, події *Closing* та *Closed* не виникають. Це важлива обставина, особливо якщо існують відкриті файли або підключення до баз даних, які потрібно очистити. Метод *Dispose* викликається обов’язково, тому, можливо, є сенс поміщати більшу частину коду очищення саме в цей метод.

Деякі властивості, що стосуються запуску форми - це *StartPosition*, *ShowInTaskbar* та *TopMost*.

StartPosition може приймати значення з переліку *FormStartPosition*:

- *CenterParent* - форма центрується в клієнтській області батьківської форми;
- *CenterScreen* - форма центрується на поточному екрані;
- *Manual* – місце розташування форми базується на властивості *Location*;

- *WindowsDefaultBounds* - форма розташовується в позиції за замовчуванням, яка визначається *Windows*, та має розміри за замовчуванням;
- *WindowsDefaultLocation* - форма розташовується в позиції за замовчуванням, яка визначається Windows, але її розміри визначаються властивістю *Size*.

Властивість *ShowInTaskbar* визначає, чи має бути форма доступна в панелі завдань (*taskbar*). Це стосується лише тих випадків, коли форма є дочірньою, а ви хочете, щоб у панелі завдань відображалась лише батьківська форма.

Властивість *TopMost* повідомляє формі, що вона повинна бути у верхній позиції *Z-порядку* програми. Це справедливо навіть для випадків, коли форма не отримує фокус негайно. Щоб користувачі могли взаємодіяти з програмою, вони мають бачити форму. Це забезпечується методами *Show* та *ShowDialog*. Метод *Show* просто робить форму видимою користувачеві. У наступному фрагменті коду показано, як створити форму та відобразити її користувачеві. Припустимо, клас форми, яку потрібно відобразити, називається *MyFormClass*:

```
MyFormClass myForm = new MyFormClass();
myForm.Show();
```

Це найпростіший шлях. Єдиний недолік полягає в тому, що код, що викликається, не отримує ніякого повідомлення про те, що форма *myForm* завершила роботу і була закрита. Іноді це не становить проблеми, і метод *Show* працюватиме нормально. Якщо ж потрібно яке-небудь повідомлення, краще використати метод *ShowDialog*.

Коли викликається метод *Show*, код, що йде за цим викликом, виконується негайно. Коли ж викликається *ShowDialog*, то код, що викликається, блокується доти, доки форма, чий метод *ShowDialog* був викликаний, не буде закрита. Але при цьому не тільки код, що викликається, блокується, але й форма необов'язково повертає значення *DialogResult*. Перелік *DialogResult* є переліком ідентифікаторів, що описують причину закриття форми. Вони включають *OK*, *Cancel*, *Yes*, *No* та ряд інших. Для того, щоб форма повернула *DialogResult*, має бути встановлена її властивість *DialogResult*, або ж властивість *DialogResult* має бути встановлена для однієї з кнопок форми.

Наприклад, припустимо, що частина програми запитує телефонний номер клієнта. Форма включає текстове поле для введення цього номера та дві кнопки: одну з позначкою *OK* та іншу - з позначкою *Cancel* (Скасувати). Якщо встановити значення властивості *DialogResult* кнопки *OK* рівним *DialogResult.OK*, а значення властивості *DialogResult* кнопки *Cancel* - *DialogResult.Cancel*, то коли будь-яка з цих кнопок буде обрана, форма стане

невидимою і поверне формі, що викликала її, відповідне значення *DialogResult*. Тепер уявімо, що форма не знищено, а лише властивість *Visible* встановлено в *false*. Це може знадобитися для того, щоб отримати інформацію від форми. Наприклад, той самий телефонний номер. За рахунок створення у формі поля для зберігання телефонного номера з'являється можливість у батьківській формі отримати це значення і викликати метод *Close* дочірньої форми. Код такої дочірньої форми може виглядати так:

```
namespace FormsSample.DialogSample
{
partial class Phone : Form
{
   public Phone()
   {
   InitializeComponent();
   btnOK.DialogResult = DialogResult.OK;
   btnCancel.DialogResult = DialogResult.Cancel;
   }

   public string PhoneNumber
   {
   get { return textBox1.Text; }
   set  { textBox1.Text = value; }
   }
      }
}
```

Перше, що необхідно відзначити – тут немає жодного коду, який обробляє натискання по кнопках. Оскільки властивість *DialogResult* встановлюється для кожної з кнопок, форма зникає з екрана після натискання або на *OK*, або на *Cancel*. Єдина додана властивість – *PhoneNumber*. У наступному коді показано метод батьківської форми, що викликає діалог *Phone*.

```
Phone frm = new Phone();
frm.ShowDialog();
if (frm.DialogResult == DialogResult.OK)
{
    label1.Text = "Номер телефона: " + frm.PhoneNumber;
}
else if (frm.DialogResult == DialogResult.Cancel)
{
     label1.Text = "Форма отменена.";
 }
frm.Close();
```

Це досить просто. Створюється новий об'єкт з ім'ям *frm*. Коли викликається метод *frm.ShowDialog()*, виконання коду батьківської форми зупиняється і чекає на повернення форми з *Phone*. Після цього можна перевірити властивість *DialogResult* форми *Phone*. Оскільки в цей момент вона ще не знищена, а тільки стала невидимою, ви, як і раніше, можете

звертатися до її загальнодоступних властивостей, однією з яких є властивість *PhoneNumber*. Після отримання даних можна викликати метод форми *Close*.

Все працює добре, але якщо повернутий номер телефону сформатований некоректно? Якщо помістити *ShowDialog* всередину циклу, можна викликати його повторно і запитувати повторне введення значення. Таким чином, зрештою отримуємо правильне значення. Не забудьте, що також потрібно обробляти *DialogResult.Cancel*, якщо користувач натискає по кнопці *Cancel*.

```
Phone frm = new Phone();
while (true)
{

   frm.ShowDialog();
   if (frm.DialogResult == DialogResult.OK)
    {
       label1.Text = "Номер телефона: " + frm.PhoneNumber;
       if (frm.PhoneNumber.Length == 8 | frm.PhoneNumber.Length == 12)
         {
            break;
         }
       else
        {
         MessageBox.Show("Неверный формат номера телефона. Исправьте. ");
        }
    }
   else if (frm.DialogResult == DialogResult.Cancel)
   {
label1.Text = "Форма отменена. ";
 break;
         }
}
frm.Close();
```

Тепер, якщо номер телефону не проходить перевірку довжини, форма *Phone* з'являється знову, щоб користувач зміг виправити помилку. Метод *ShowDialog* не створює нового екземпляра форми. Будь-який текст, введений у форму, залишиться в ній, тому якщо форма має бути очищена, це має зробити сам користувач.

3.2. Зовнішній вигляд

Перше, що бачить користувач – це форма програми. Вона має з'являтися першою та бути максимально функціональною. Якщо програма не вирішує жодних бізнес-завдань, то насправді не важливо, як вона виглядає. Це не означає, однак, що форма та весь дизайн графічного інтерфейсу користувача не повинні бути приємними на вигляд. Такі прості речі, як комбінація кольорів, розміри шрифтів та вікон можуть значно полегшити роботу користувача. Іноді не потрібно, щоб користувач мав доступ до системного

меню. Мається на увазі меню, яке з'являється при натисканні по піктограмі, розташованій у верхньому лівому куті вікна. Зазвичай воно містить такі елементи, як *Restore*, *Minimize*, *Maximize* та *Close*. Властивість *ControlBox* дозволяє встановити видимість системного меню. Видимість кнопок *Minimize* та *Maximize* можна задати властивостями *MaximizeBox* та *MinimizeBox*. Якщо видалити всі кнопки і потім привласнити властивості *Text* порожній рядок (*""*), смуга заголовка зникне повністю.

Якщо встановити властивість *Icon* форми та не встановити *ControlBox* у *false*, то у лівому верхньому куті з'явиться піктограма. Зазвичай її встановлюють у *app.ico*. Це надає кожній формі піктограму, що збігається з піктограмою програми.

Властивість *FormBorderStyle* встановлює тип рамки, що оточує форму через значення з переліку *FormBorderStyle*:

- *Fixed3D*
- *FixedDialog*
- *FixedSingle*
- *FixedToolWindow*
- *None*
- *Sizable*
- *SizableToolWindow*

Більшість наведених вище значень не потребує пояснень, крім двох рамок інструментальних (*Tool*) вікон. Вікна *Tool* не з'являються на панелі завдань, незалежно від установки *ShowInTaskBar*. Крім того, такі вікна не відображаються у списку вікон, коли натискається *<Alt+Tab>*. За замовчуванням встановлено тип рамки *Sizeable*.

Якщо тільки не висунуто спеціальних вимог, кольори більшості елементів графічного інтерфейсу користувача повинні бути встановлені системними, а не конкретними. При цьому якщо який-небудь користувач вважає за краще мати зелені кнопки з червоним текстом, він може це налаштувати для всієї системи і додаток прийме такі налаштування. Щоб встановити для елемента керування певний системний колір, потрібно викликати метод *FromKnownColor* класу *System.Drawing.Color*. Цей метод набуває значення з переліку *KnownColor*. Багато кольорів визначено у цьому перерахуванні як кольори різних елементів графічного інтерфейсу користувача - наприклад, *Control*, *ActiveBorder* та *Desktop*. Тому якщо, наприклад, колір фону форми повинен завжди відповідати кольору *Desktop*, потрібно застосувати наступний код:

```
myForm.BackColor = Color.FromKnownColor(KnownColor.Desktop);
```

Тепер, якщо користувач змінить колір робочого стола, колір фону форми зміниться разом з ним. Це правильний, дружній підхід до налаштування

зовнішнього вигляду програми. Іноді користувачі вибирають досить дивні поєднання кольорів для своїх робочих столів, однак, це їх вибір.

У *Windows XP* було введено засіб під назвою візуальних стилів. Візуальні стилі змінюють спосіб відображення кнопок, текстових полів, меню та інших елементів керування, а також вид курсора миші.

Дозволити візуальні стилі для своїх програм можна за допомогою методу *Application.EnableVisualStyles*. Цей метод має бути викликаний перед створенням елементів графічного інтерфейсу будь-якого роду. З цієї причини він зазвичай викликається в методі *Main*, як показано в наступному прикладі:

```
[STAThread] static void Main()
{
  Application.EnableVisualStyles();
  Application.Run(new Form1());
}
```

Цей код дозволяє багатьом елементам керування, які підтримують візуальні стилі, скористатися їх перевагами. Через наслідки застосування методу *EnableVisualStyles()* відразу після його виклику може знадобитися викликати метод *Application.DoEvents()*. Це вирішить проблему зникнення піктограм та панелей інструментів під час виконання. До того ж метод *EnableVisualStyles* доступний лише у *.NET Framework 1.1*.

Вам доведеться вирішувати ще одне завдання, що стосується елементів керування. Більшість елементів керування має властивість *FlatStyle*, що приймає значення з переліку *FlatStyle*:

- *Flat – аналогічно flat, за винятком того, що коли курсор миші знаходиться на елементі керування, він набуває тривимірної форми;*
- *Standard – елемент керування має тривимірну форму;*
- *System – зовнішній вигляд елемента керування керується системою.*

Щоб увімкнути візуальні стилі, властивість *FlatStyle* елемента керування повинна бути встановлена рівною *FlatStyle.System*. Після цього додаток буде мати зовнішній вигляд *XP* та підтримувати теми *XP*.

3.3. Багатодокументний інтерфейс (MDI)

Програми в стилі *MDI* використовуються, коли потрібно відображати або множину екземплярів однотипних форм, або різних форм, які повинні бути включені до програми однаковим чином. Прикладом безлічі екземплярів однотипних форм може бути текстовий редактор, що одночасно відображає безліч вікон редагування. Прикладом додатків другого типу може бути *Microsoft Access*. У ньому можна одночасно мати відкритими вікна запитів, вікна дизайнера та вікна таблиць. Всі ці вікна ніколи не виходять за межі головного вікна *Access*.

Проект, що містить приклади цієї глави, сам є прикладом *MDI-додатку*. Форма *mdiParent* у цьому проекті є батьківською формою *MDI*. Встановлення властивості *IsMdiContainer* значення *true* робить будь-яку форму батьківської форми *MDI*. Якщо форма знаходиться у дизайнері, це можна відразу побачити, оскільки її колір змінюється на темно-сірий. Це свідчить, що ця форма стала батьківської формою *MDI*. До неї можна додавати елементи керування, але зазвичай це не рекомендується.

Для того, щоб форма поводилася як дочірня форма *MDI*, вона повинна знати, яка форма є для неї батьківською. Це робиться присвоєнням властивості *MdiParent* посилання на батьківську форму. У прикладі всі дочірні форми створюються методом *ShowMdiChild*, що приймає посилання на дочірню форму, яка повинна бути показана. Після встановлення властивості *MdiParent* у *this*, тобто. посилання на форму *mdiParent*, дочірня форма відображається. Код методу *ShowMdiChild* виглядає так:

```
private void ShowMdiChild(Form childForm)
 {
   childForm.MdiParent = this;
   childForm.Show();
}
```

Одна з особливостей *MDI-додатків* полягає в тому, що будь-якої миті може бути відкрито декілька дочірніх форм. Посилання на поточну активну дочірню форму можна отримати з властивості *ActiveMdiChild* батьківської форми. Це показує пункт *Windows-меню Current Active*. Якщо його обрати, то буде відображено вікно повідомлення з ім’ям форми та текстовим значенням.

Дочірні форми можуть бути розміщені викликом методу *LayoutMdi*. Він приймає параметр типу перерахування *MdiLayout*, можливі значення якого включають *Cascade*, *TileHorizontal* та *TileVertical.*

3.4. Елементи керування на замовлення

Застосування елементів керування та компонентів - це те, що робить розробку за допомогою такого пакета, як *Windows Forms*, дуже продуктивною. Можливість створювати власні елементи керування та компоненти робить її ще більш продуктивною. Створюючи елементи керування, можна інкапсулювати функціональність у пакети, які можуть повторно використовуватися знову і знову.

Елементи керування створюються у різний спосіб. Можна розпочати з нуля, успадкувавши новий клас від *Control*, *ScrollableControl* або *ContainerControl*. При цьому потрібно буде перевизначити подію Paint, щоб повністю виконувати все малювання, не кажучи вже про додавання функціональності, яку елемент керування повинен забезпечувати. Якщо елемент керування задуманий як розширена версія деякого існуючого, то потрібно оголосити його спадкоємцем класу елемента керування, що розширюється. Наприклад, якщо необхідний елемент керування *TextBox*,

який змінюватиме колір фону при встановленні властивості *ReadOnly*, то повне створення абсолютно нового *TextBox* буде марною тратою часу. У цьому випадку успадкуйте новий клас від *TextBox* і перевизначте властивість *ReadOnly*. Оскільки властивості *ReadOnly* класу *TextBox* не позначені словом *override*, в оголошенні потрібно використовувати конструкцію *new*. У наступному коді показано приклад такого перевизначення властивості *ReadOnly*:

```
public new bool ReadOnly
{
get { return base.ReadOnly;}
set
{
if (value) this.BackgroundColor = Color.Red;
else this.BackgroundColor = Color.FromKnowColor(KnownColor.Window);
base.ReadOnly = value;
 }
}
```

Тут бачимо, що *get* повертає те, що й базовий об'єкт. Опитування властивості не пов'язане з нашим завданням зміни кольору фону при встановленні *ReadOnly*, тому ми просто передаємо функціональність базовому об'єкту. При встановленні властивості слід перевірити передане нове значення - *false* або *true*. Якщо воно ріне *true*, то змінюємо колір (в даному випадку на червоний), а якщо *false* - встановлюємо *BackgroundColor* у колір за замовчуванням. Після цього значення передається базовому об'єкту, щоб текстове поле дійсно стало доступним відповідно до значення параметра. Як бачимо, перевизначення однієї простої властивості дозволяє додати нову функціональність елементу керування. Елементу керування на замовлення можна додати атрибути, які розширюють можливості часу проектування. У табл. 3.1. описані деякі з найчастіше використовуваних атрибутів.

Таблиця 3.1

Часто використовувані атрибути елементів керування на замовлення

Ім'я атрибута	Опис
BindableAttribute	Використовується під час проектування для визначення того, чи підтримує властивість двосторонню прив'язку даних.
BrowsableAttribute	Визначає, чи має властивість відображатись у візуальному дизайнері.
CategoryAttribute	Визначає, під якою категорією властивість відображається у вікні властивостей. Використовує певні категорії або дозволяє створювати нові. За замовчуванням - Misc.
DefaultEventAttribute	Специфікує стандартну подію для класу.
DefaultPropertyAttribute	Специфікує стандартну властивість для класу.
DefaultValueAttribute	Специфікує значення за замовчуванням для властивості. Зазвичай, це початкове значення.

Продовження таблиці 3.1

Ім’я атрибута	Опис
DecriptionAttribute	Текст, що з’являється у нижній частині вікна дизайнера при виборі властивості.
DesignOnlyAttribute	Позначає властивість як таку, яка може редагуватися лише у режимі проектування.

Доступні також інші атрибути, що стосуються редактора, який використовується в процесі проектування, а також інші розширені можливості часу проектування. Майже завжди мають бути додані атрибути *Category* та *Description*. Це допомагає іншим розробникам, які використовують елемент керування, краще розуміти призначення властивості. Щоб додати підтримку засобу *IntelliSense*, слід додавати *XML - коментар* до кожної властивості, методу та події. Коли елемент керування компілюється з опцією */doc*, згенерований *XML-файл* коментарів представляє елементу керування підтримку *IntelliSense*.

3.5. Елемент керування на замовлення базуючись на TreeView

У цьому розділі ми продемонструємо розробку елемента керування на базі стандартного *TreeView*. Він відображатиме файлову структуру дискового пристрою. Ми додамо властивості, які встановлюють базову або кореневу папку та визначають, як мають відображатися папки та файли. Ми також скористаємося різними атрибутами, про які йшлося у попередньому розділі. Як і для будь-якого іншого проекту, необхідно визначити вимоги до нового елементу керування. Ось список базових вимог, які потрібно буде реалізувати:

- читати папки та файли та відображати їх користувачу;
- відображати структуру папок у деревоподібному ієрархічному поданні;
- необов’язково приховувати файли від подання;
- визначати папку, яка буде базовою кореневою папкою;
- повертати поточну вибрану папку;
- надавати можливість відкласти завантаження файлової структури.

Це може бути гарною початковою точкою. Одна з вимог задовольняється лише фактом успадкування нового елемента керування від *TreeView*.

Елемент керування TreeView відображає дані в ієрархічному вигляді. Він показує текст, що описує об’єкт у списку, та необов’язково – піктограму. Список може відкриватися та закриватися натисканням по об’єкту або натисканням клавіш зі стрілками. Створимо у *Visual Studio .NET* новий проект *Windows Control Library* з ім'ям *FolderTree* та видалимо з нього клас

UserControl1. Додамо новий клас та назвемо його *FolderTree*. Оскільки *FolderTree* успадковуватиметься від *TreeView*, змінимо оголошення класу:

public class FolderTree

на

public class FolderTree : System.Windows.Forms.TreeView

У цей момент ми вже отримуємо повністю функціональний елемент керування *FolderTree*. Він робить все, що може робити *TreeView*, але не більше того. Елемент керування *TreeView* підтримує колекцію об'єктів *TreeNode*. Ми не можемо завантажувати файли та папки безпосередньо в цей елемент керування. Існує декілька можливостей відобразити вузли *TreeNode*, які завантажуються в колекцію *Nodes*, що належить *TreeView*, на папки та файли, які вони представляють. Наприклад, коли обробляється кожна папка, створюється новий об'єкт *TreeNode*, і його властивість *text* набуває значення імені папки чи файла. Якщо в якийсь момент знадобиться деяка додаткова інформація про файл або папку, можна виконати ще одне звернення до диска, щоб отримати цю інформацію, або зберегти додаткові дані щодо файлу або папки у властивості *Tag*. Інший метод полягає у створенні нового класу-спадкоємця *TreeNode*. Додатково до його базової функціональності можна додати низку нових методів та властивостей. Саме таким шляхом ми підемо в цьому прикладі. Це забезпечить можливість більш гнучкого дизайну. Якщо потрібні нові властивості, їх можна легко додавати, не торкаючись існуючого коду. У елемент керування потрібно завантажувати два типи об'єктів: папки та файли. Кожен з них має свої власні характеристики. Наприклад, кожна папка включає об'єкт *DirectoryInfo*, який містить додаткову інформацію, а файл - об'єкт *FileInfo*. Через цю відмінність ми будемо використовувати два різних класи для завантаження елемента керування *TreeView*: *FileNode* та *FolderNode*. Додамо ці два класи до проекту, при цьому успадкуємо кожен із них від *TreeNode*. Нижче наведено код *FileNode*.

```
namespace FormsSample.SampleControls
{
   public class FileNode : System.Windows.Forms.TreeNode
 {
string _fileName = "";
FileInfo _info; public FileNode(string fileName)
{
_fileName = fileName; _
info = new FileInfo(_fileName);
base.Text = _info.Name;
if (_info.Extension.ToLower() == ".exe") this.ForeColor =
System.Drawing.Color.Red;
}

public string FileName
```

```
{
get { return _fileName; }
set { _fileName = value; }
}
public FileInfo FileNodeInfo
{
 get { return _info; }
 }
    }
}
```

Ім’я файлу, що обробляється, передається конструктору *FileNode*. У конструкторі створюється об’єкт *FileInfo* для файлу і надається змінній-члену *_info*. Властивість *base.Text* встановлюється рівною імені файлу. Оскільки ми успадковуємо цей клас *TreeNode*, то за допомогою цього встановлюється властивість *Text* класу *TreeNode*. Це текст, який буде відображено в елементі керування *TreeView*.

Далі додаються дві властивості для отримання даних. *FileName* поверне ім’я файлу, а *FileNodeInfo* – об’єкт *FileInfo*, який описує файл.

Нижче наведено код класу *FolderNode*. Він дуже схожий на структуру класу *FileNode*. Різниця полягає в тому, що замість властивості *FileInfo* тут представлена властивість *DirectoryInfo*, а замість *FileName – FolderPath*.

```
namespace FormsSample.SampleControls
{
 public class FolderNode : System.Windows.Forms.TreeNode
{
        string _folderPath = "";
        DirectoryInfo _info;

        public FolderNode(string folderPath)
        {
        folderPath = folderPath;
        info = new DirectoryInfo(folderPath); this.Text = _info.Name;
        }

public string FolderPath
{
 get { return _folderPath; }
 set { _folderPath = value; }
}
public DirectoryInfo FolderNodeInfo
{
   get { return _info; }
}
}
}
```

Тепер можна конструювати елемент керування *FolderTree*. Відповідно до вимог нам знадобиться властивість для читання та ініціалізації *RootFolder*. Також знадобиться властивість *ShowFiles* для визначення того, чи мають відображатися файли у дереві. Властивість *SelectedFolder* поверне поточну виділену папку у дереві. Таким чином, код елемента керування *FolderTree* буде виглядати так, як показано нижче:

```
using System;
using System.Windows.Forms;
using System.IO;
using System.ComponentModel;

namespace FolderTree
{
/// <summary>
/// Summary description for FolderTreeCtrl.
/// </summary>

public class FolderTree :: System.Windows.Forms.TreeView
{
  string _rootFolder = "";
  bool _showFiles = true;
  bool _inInit = false;

public FolderTree() { }

 [Category("Behavior"),
Description("Gets or sets the base or root folder of the tree"),
DefaultValue("C:\\ ")]

public string RootFolder
{
get {return _rootFolder;}
set
{
    _rootFolder = value;
    if(!_inInit) InitializeTree();
 }
}
[Category("Behavior"), Description("Indicates whether files will be seen in the
list. "), DefaultValue(true)]

public bool ShowFiles
 {
    get{return _showFiles;}
    set{showFiles = value;}
 }
 [Browsable(false)]

public string SelectedFolder
{
```

```
get
        {
          if(this.SelectedNode is FolderNode) return ((Folder
          Node) this.SelectedNode).FolderPath;
          return "";
        }
}
}
}
```

Тут додано три властивості: *ShowFiles*, *SelectedFolder* та *RootFolder*. Зверніть увагу на атрибути цих властивостей. Ми встановили *Category*, *Description* та *DefaultValues* для *ShowFiles* та *RootFolder*. Ці дві властивості з'являться у браузері властивостей у режимі проектування. У *SelectedFolder* насправді немає сенсу під час проектування, тому йому встановлено атрибут *Browsable = false*. *SelectedFolder* не з'явиться у браузері властивостей, однак, оскільки це загальнодоступна властивість, вона з'явиться в *IntelliSense* і буде доступною в коді.

Далі слід ініціалізувати завантаження файлової системи. Ініціалізація елемента керування може бути непростою. Ініціалізація – як під час проектування, так і під час виконання – має бути добре продуманою. Коли елемент керування встановлюється у конструкторі, він насправді запускається. Якщо, наприклад, у конструкторі є звернення до бази даних, це звернення виконається, коли ви перетягнете елемент керування у полі конструктора. У випадку *FolderTree* це може спричинити певні наслідки.

Ось як виглядатиме метод, який насправді завантажує файли:

```
private void LoadTree(FolderNode folder)
 {
 string[] dirs = Directory.GetDirectories(folder.FolderPath);
foreach(string dir in dirs)
{
   FolderNode tmpfolder = new FolderNode(dir);
   folder.Nodes.Add(tmpfolder);
   LoadTree(tmpfolder);
}

if(_showFiles)
{
    string[] files = Directory.GetFiles(folder.FolderPath);
    foreach(string file in files)
            {
 FileNode fnode = new FileNode(file);
 folder.Nodes.Add(fnode);
            }
 }
 }
```

showFiles – змінна-член класу логічного типу, яка встановлюється через властивість *ShowFiles*. Якщо вона дорівнює *true*, файли відображаються у дереві. Єдине питання – коли слід викликати *LoadFiles*? Існує кілька варіантів. Цей метод може бути викликаний під час встановлення властивості *RootFolder*. У деяких ситуаціях це бажано, але не під час проектування. Згадаймо, елементи керування в дизайнері "живі", тому коли встановлюватиметься властивість *RootFolder*, наш елемент керування спробує виконати завантаження з файлової системи. Вирішити цю проблему можна, перевіривши властивість *DesignMode*. Воно повертає true, якщо елемент керування перебуває зараз у дизайнері. Тепер можна записати код ініціалізації так:

```
private void InitializeTree()
{
if(!this.DesignMode)
{
FolderNode rootNode = new FolderNode(_rootFolder);
LoadTree(rootNode); this.Nodes.Clear();
this.Nodes.Add(rootNode);
}
}
```

Якщо елемент керування не перебуває в режимі проектування, і *rootFolder* не дорівнює порожньому рядку, почнеться завантаження дерева. Насамперед створюється і передається методу *LoadTree* кореневий вузол *Root*. Інший спосіб полягає у тому, щоб реалізувати загальнодоступний метод *Init*. У методі *Init* можна здійснити виклик *LoadTree*. Проблема такого способу пов'язана з тим, що розробник, який використовує елемент керування, має викликати метод *Init*. Але в деяких випадках це може виявитися прийнятним рішенням. Для додаткової гнучкості можна реалізувати інтерфейс *ISupportInitialize*. Цей інтерфейс має два методи - *BeginInit* та *EndInit*. Коли елемент керування реалізує *ISupportInitialize*, методи *BeginInit* та *EndInit* викликаються автоматично у згенерованому коді *InitializeComponent*. Це дозволяє відкласти процес ініціалізації до того моменту, коли будуть встановлені всі властивості.

ISupportInitialize дозволить коду в батьківській формі також відкласти ініціалізацію. Якщо в коді буде встановлено властивість *RootNode*, то виклик *BeginInit* дозволить спочатку встановити властивість *RootNode*, як і всі інші властивості, або виконати необхідні дії перш, ніж наш елемент керування завантажить файлову систему. Коли викликається *EndInit*, відбудеться ініціалізація. Ось як можуть виглядати *BeginInit* та *EndInit*:

```
#region ISupportInitialize Members
public void ISupportInitialize.BeginInit()
{
    _inInit = true;
}
```

```
public void ISupportInitialize.EndInit()
{
    if(_rootFolder != "")
{
     InitializeTree();
}
_inInit = false;
}
#endregion
```

Все, що робиться в методі *BeginInit* - це присвоєння змінній *_inInit* значення *true*. Цей прапор застосовується для того, щоб визначити, що елемент керування знаходиться в процесі ініціалізації, і використовується як *RootFolder*. Якщо властивість *RootFolder* встановлюється поза класом *InitializeComponent*, це означає, що дерево потребує повторної ініціалізації. У властивості *RootFolder* ми перевіряємо, чому дорівнює *_inInit* - *true* або *false*. Якщо *true*, це означає, що нам не потрібно проходити через процес ініціалізації. Якщо ж прапор *_inInit* дорівнює *false*, ми викликаємо *InitializeTree*. Можна також мати загальнодоступний метод *Init* і в ньому виконувати те саме завдання.

У методі *EndInit* ми перевіримо, чи знаходиться наш елемент керування в режимі проектування і що з *_rootFolder* асоційовано коректний шлях. Тільки після цього викликається *InitializeTree*.

Щоб надати нашому елементу керування професійного вигляду, необхідно додати бітове зображення. Це буде піктограма, що відображається на панелі інструментів, коли елемент керування буде додано до проекту. Бітове зображення має бути розміром 16×16 пікселів та мати 16 кольорів. Файл цього зображення можна створити в будь-якому графічному редакторі, що дозволяє витримати такий розмір і глибину кольору. Це можна зробити навіть у *Visual Studio .NET*. Для цього потрібно натиснути правою кнопкою миші по проекту та вибрати в контекстному меню пункт *Add New Item* (Додати новий елемент). Виберіть *Bitmap File* (Файл бітового зображення), щоб відкрити графічний редактор. Після створення файлу бітового зображення додайте його до проекту, переконавшись, що він знаходиться в тому самому просторі імен і з тим самим ім’ям, як у нашого елемента керування.

І, нарешті, встановіть властивість *BuildAction* для цього ресурсу *Embedded Resource*: натисніть правою кнопкою миші по файлу зображення у *провіднику рішень* (Solution Explorer) і виберіть в контекстному меню пункт *Properties* (Властивості). Виберіть *Embedded Resource* для властивості *BuildAction*.

Щоб протестувати розроблений елемент керування, створимо у тому ж рішенні проект *TestHarness*. Це має бути простий додаток *Windows Forms* з єдиною формою. До розділу посилань додамо посилання на проект *FolderTreeCtl*. У вікні *Toolbox* додамо посилання *FolderTreeCtl.DLL*. Після

цього *FolderTreeCtl* повинен з'явитися на панелі інструментів з доданим раніше зображенням у вигляді піктограми.

Натисніть по піктограмі і перетягніть її на форму *TestHarness*. Встановимо *RootFolder* на одну з доступних папок та запустимо рішення.

Це, безперечно, повноцінний та новий елемент керування. Але ще дещо можна зробити для того, щоб він був більш повнофункціональним та готовим до промислового застосування. Наприклад, йому можна додати наведені нижче всластивості.

- *Винятки* - якщо елемент керування намагається завантажити папку, до якої користувач не має доступу, повинно виникати виключення.
- *Фонове завантаження* – завантаження великого дерева папок може вимагати значного часу. Тому можна удосконалити процес ініціалізації так, щоб скористатися перевагами багатопоточності для фонового завантаження.
- *Кодування за допомогою кольору* - можна виводити імена файлів певних типів у різних кольорах.
- *Піктограми* - можна додати елемент керування *ImageList*, до якого можна помістити піктограми кожного завантаженого файлу або папки.

3.6. Користувальницькі елементи керування

Користувальницькі елементи керування – один із найпотужніших засобів *Windows Forms*. Вони дозволяють інкапсулювати інтерфейс користувача в симпатичні повторно використовувані пакети, які можна легко підключати до різних проектів. Нерідко організації мають набір бібліотек з часто використовуваними елементами керування власної розробки. Користувальницькі елементи керування можуть містити не тільки функціональність інтерфейсу користувача, але також деякі загальні функції перевірки достовірності даних - такі як формат телефонних номерів або номерів ідентифікаторів. Користувальницькі елементи керування можуть мати вбудовані списки елементів, що застосовуються для швидкого завантаження в інтерфейсні елементи - вікна списків і комбіновані списки. Коди штатів та країн також входять до цієї категорії. Включення якомога більшого обсягу функціональності, яка залежить від конкретного додатка, в елементи керування дозволяє зробити їх більш корисними для цієї організації. У цьому розділі ми створимо простий елемент керування для введення адреси. Ми також додамо різні події, які забезпечать можливість прив'язування даних. Цей елемент керування буде складатися з текстових полів для введення двох рядків адреси, міста, штату та поштового коду. Щоб створити власний елемент керування в поточному проекті, натисніть правою кнопкою миші по *Solution Explorer* і виберіть в контекстному меню пункт *Add ->Add New User Control* (Додати ->Додати новий елемент керування).

Можна також створити новий проект *Control Library* (Бібліотека елемента керування) та додати до нього цей елемент керування. Після того, як новий елемент керування буде запущений, ми побачимо в дизайнері форму без рамок. Сюди ми помістимо елементи керування, з яких складатиметься наш елемент користувача. Нагадаємо, що користувальницький елемент керування складається з одного або більше елементів керування, доданих у контейнер, тому все це нагадує створення форми. Для введення адреси нам знадобиться п’ять текстових полів (*TextBox*) та три написи (*Label*). Розташувати їх можна будь-яким відповідним чином, як показано на рис. 3.1.

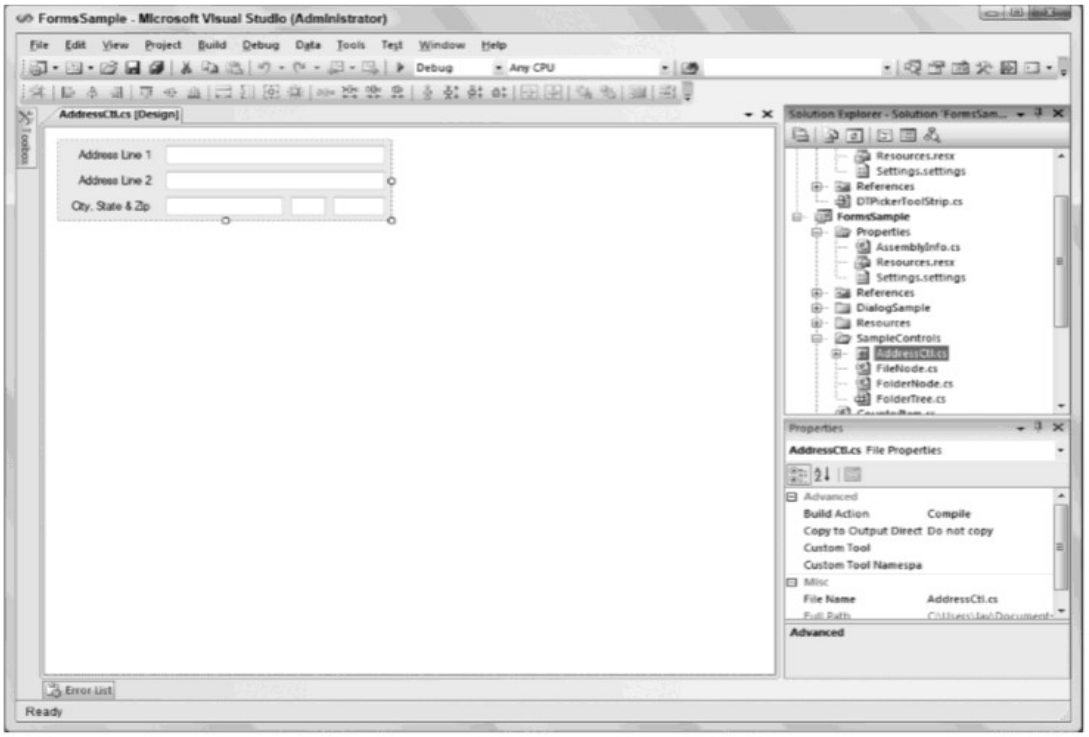


Рис. 3.1. Приклад компонування користувальницького елемента керування

Імена текстових полів у нашому прикладі будуть такими:

- *txtAddress1*
- *txtAddress2*
- *txtCity*
- *txtState*
- *txtZip*

Після того, як текстові поля буде розміщено та отримані коректні імена, додамо загальнодоступні властивості. У вас може виникнути спокуса зробити складові текстові поля загальнодоступними (*public*), а не приватними (*private*). Однак, це погана ідея, оскільки порушується принцип інкапсуляції функціональності, яку ви можете побажати додати до властивостей.

Нижче наведено код властивостей, які слід додати до нашого елементу керування.

```
public string AddressLine1
{
    get{return txtAddress1.Text;}
set
{
if(txtAddress1.Text != value)
{
  txtAddress1.Text = value;
if(AddressLine1Changed != null) AddressLine1Changed(this, EventArgs.Empty);
 }
}
}

public string AddressLine2
{

get{return txtAddress2.Text;}
set
{
     if(txtAddress2.Text != value)
     {
       txtAddress2.Text = value;
if(AddressLine2Changed != null) AddressLine2Changed(this, EventArgs.Empty);
      }
 }
       }

public string City
{
get{return txtCity.Text;}
set
{
if(txtCity.Text != value)
{
 txtCity.Text = value;
 if(CityChanged != null) CityChanged(this, EventArgs.Empty);
 }
 }
 }

public string State
{
get{return txtState.Text;}
set
{
if(txtState.Text != value)
{
txtState.Text = value;
if(StateChanged != null) StateChanged(this, EventArgs.Empty);
 }
}
}
```

```
public string Zip
{
get{return txtZip.Text;}
set
{
if(txtZip.Text != value)
{
  txtZip.Text = value;
  if(ZipChanged != null) ZipChanged(this, EventArgs.Empty);
}
}
}
```

Екземпляри властивості *get* досить прямолінійні. Вони повертають значення відповідних текстових властивостей елемента керування *TextBox*. Екземпляри властивості *set*, навпаки, виконують трохи більше роботи. Усі вони функціонують однаково. Спочатку виконується перевірка того, чи змінюється значення властивості. Якщо нове значення збігається зі старим, нічого не відбувається. Якщо ж нове значення відрізняється, воно привласнюється текстовій властивості елемента *TextBox*, після цього перевіряється, чи існує екземпляр події. Йдеться про події зміни властивостей, які мають спеціальний формат імені - *propertynameChanged*, де *propertyname* – ім’я властивості. По відношенню до властивості *AddressLine1* подія називається *AddressLine1Changed*. Події оголошуються так, як показано нижче:

```
public event EventHandler AddressLine1Changed;
public event EventHandler AddressLine2Changed;
public event EventHandler CityChanged;
public event EventHandler StateChanged;
public event EventHandler ZipChanged;
```

Призначення цих подій - повідомляти прив’язку про те, що властивість змінилася. Виконавши перевірку допустимості, прив’язка забезпечить синхронізацію нового значення властивості з об’єктом, до якого вона прив’язана. Ще один крок має бути виконано для підтримки прив’язки. Зміна текстового поля, виконана користувачем, не встановить нове значення властивості безпосередньо. Тому необхідно також викликати подію *propertynameChanged* при зміні значення в текстовому полі. Найпростіший спосіб зробити це - відслідковувати подію *TextChanged* елемента керування *TextBox*. У нашому прикладі буде тільки один обробник подій *TextChanged* і всі текстові поля будуть використовувати його. Ім’я елемента керування перевіряється, щоб побачити, в якому саме елементі відбулася зміна, а потім викликати відповідну подію *propertyNameChanged*. Нижче наведено код цього обробника подій.

```
private void controls_TextChanged(object sender, System.EventArgs e)
{
switch(((TextBox)sender).Name)
{
case "txtAddress1" :
if(AddressLine1Changed != null) AddressLine1Changed(this, EventArgs.Empty);
break;
case "txtAddress2" :
if(AddressLine2Changed != null) AddressLine2Changed(this, EventArgs.Empty);
break;
case "txtCity" :
if(CityChanged != null) CityChanged(this, EventArgs.Empty);
break;
case "txtState" :
if(StateChanged != null) StateChanged(this, EventArgs.Empty);
break;
case "txtZip" :
if(ZipChanged != null) ZipChanged(this, EventArgs.Empty);
break;
 }
}
```

Тут ми застосовуємо простий оператор *switch* для визначення того, яке текстове поле викликало подію *TextChanged*. Після цього виконується перевірка, щоб переконатися, що подія коректна і не дорівнює *null*.

Потім виникає подія *Changed*. Зверніть увагу, що при цьому відправляються порожні *EventArgs* (*EventArgs.Empty*). Той факт, що ці події були додані до властивостей для підтримки прив'язування даних, не означає, що єдиний спосіб використання нашого елемента керування проходить через прив'язку даних. Їх додано так, щоб користувальницький елемент керування міг використовувати прив'язку у разі її наявності. Це лише один із способів забезпечення максимальної гнучкості користувальницького елемента керування.

Пам'ятаючи про те, що користувальницький елемент керування - це, по суті, елемент керування з додатковими можливостями, все пов'язане із застосуванням його в середовищі дизайнера, що ми обговорювали в попередньому розділі, також може бути застосовано й тут. Ініціалізація користувальницького елемента керування може призвести до тих самих наслідків, що ми бачили в прикладі *FolderTree*. Під час проектування елементів керування необхідно потурбуватися про те, щоб уникнути звернення до сховищ даних, які можуть бути недоступними розробникам, які використовують ваші елементи керування.

Ще одна особливість, подібна до створення елементів керування - це атрибути, що застосовуються до користувальницьких елементів керування. Загальнодоступні властивості та методи користувальницького елемента керування відображаються у вікні властивостей, коли він поміщається в дизайнер. У прикладі з адресою непогано було б додати атрибути *Category*, *Description* і *DefaultValue* до властивостей адреси. Можна створити нову

категорію *AddressData* зі значенням за замовчуванням "". Нижче наведено приклад застосування цих атрибутів до властивості AddressLine1.

```
 [Category("AddressData"), Description("Gets or sets the AddressLine1 value"),
DefaultValue("")]

public string AddressLine1
{
get{return txtAddress1.Text;}
set
{
if(txtAddress1.Text != value)
    {
txtAddress1.Text = value;
if(AddressLine1Changed != null) AddressLine1Changed(this, EventArgs.Empty);
    }
}
}
```

Як бачите, все, що потрібно зробити для додавання нової категорії – це встановити текст в атрибуті *Category*. Це автоматично додає нову категорію. Крім того, що було описано вище, залишається великий простір для вдосконалення. Наприклад, можна увімкнути список назв штатів та їх скорочень. Замість однієї властивості штату користувальницький елемент керування можна розширити властивостями, що дозволяють вводити і зберігати як ім'я штату, так і його абревіатуру. Також можна додати обробку винятків. Можна подумати про те, чи має властивість *AddressLine2* бути необов'язковою, куди слід вводити номер квартири та кімнати тощо.

У наведеному вище матеріалі було дано уявлення про базові принципи побудови клієнтських *Windows-додатків*. Були описані стандартні елементи керування, що утворюють ієрархію класів простору імен *Windows.Forms*, і розглянуто різні властивості та методи. Також тут було продемонстровано, як можна створити базовий елемент керування на замовлення, а також базовий користувальницький елемент керування. Потужність та гнучкість, що забезпечується ними, переоцінити неможливо. Завдяки можливостям створення власних наборів користувальницьких елементів керування, значно полегшується завдання розробки та тестування клієнтських *Windows-додатків*, оскільки знову і знову можна використовувати одні і ті ж ретельно протестовані компоненти. У наступному розділі наводяться тексти та опис програм, в основу яких закладено використання елементів керування.

4. Приклади використання елементів керування

Елементи керування є візуальними класами, які отримують введені користувачем дані і можуть ініціювати різні події. Всі елементи керування успадковуються від класу *Control* і тому мають низку загальних властивостей:

- *Anchor*: Визначає, як елемент розтягуватиметься
- *BackColor*: Визначає фоновий колір елемента
- *BackgroundImage*: Визначає фонове зображення елемента
- *ContextMenu*: Контекстне меню, яке відкривається, натиснувши по елементу правою кнопкою миші. Задається за допомогою елемента ContextMenu
- *Cursor*: Показує, як відображатиметься курсор миші при наведенні на елемент
- *Dock*: Задає розташування елемента на формі
- *Enabled*: Визначає, чи доступний елемент для використання. Якщо ця властивість має значення *False*, елемент блокується.
- *Font*: Встановлює шрифт тексту для елемента
- *ForeColor*: Визначає колір шрифту
- *Location*: Визначає координати верхнього лівого кута елемента керування
- *Name*: Ім’я елемента керування
- *Size*: Визначає розмір елемента
- *Width*: ширина елемента
- *Height*: висота елемента
- *TabIndex*: Визначає порядок обходу елемента, натиснувши клавішу *Tab*
- *Tag*: Дозволяє зберігати значення, асоційоване з цим елементом керування

4.1. Button, Label

Приклад 1.

У програмі *examp01*, наведеній у лістингу 4.1., елемент керування *"кнопка"* створюється під час створення об’єкта *Button*. Кнопка повідомляє про те, що її натиснули, використовуючи встановлений для неї обробник події *Click*. Ця програма створює єдину кнопку, при натисканні якої форма виводить текст *"Button cliclcked!"*. Через секунду текс виводиться в "червоній" прямокутній ділянці, що обрамляє цей текст. Ще через секунду текст зникає і не виділяється червона прямокутна область, оскільки ця область фарбується кольором форми. Результат виконання програми наведено на рис. 4.1.

Лістинг 4.1.(examp01)

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System. Threading;

namespace examp01
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            Text = "Simple Button";
            button1.Parent = this;
            button1.Text = "Click Mel";
            button1.Location = new Point(100,100);
        }

        private void button1_Click(object sender, EventArgs e)
        {
           Graphics gr = CreateGraphics();
           Point pt = Point.Empty;
           string str = "Button clicked!";
           gr.DrawString(str, Font, new SolidBrush(ForeColor), pt);
           Thread.Sleep(1000);
           gr.FillRectangle(new SolidBrush(Color.FromArgb(255,0,0)),
           new RectangleF(pt,gr.MeasureString(str, new Font("Times new
           Roman",20F))));
        }
    }
}
```

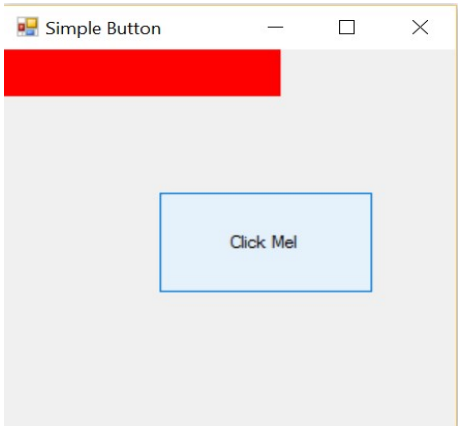


Рис. 4.1. Обробка подій при натисканні кнопки

Приклад 2.

У цьому прикладі дві кнопки використовуються для демонстрації роботи з розмірами кнопок. Натискання однієї кнопки робить форму на 10% менше, інший - на 10% більше (у межах обмежень, що накладаються Windows). Кнопки залишаються у центрі клієнтської області. Ці кнопки показано на рис. 4.2. У лістингу 4.2. показано реалізацію програми.

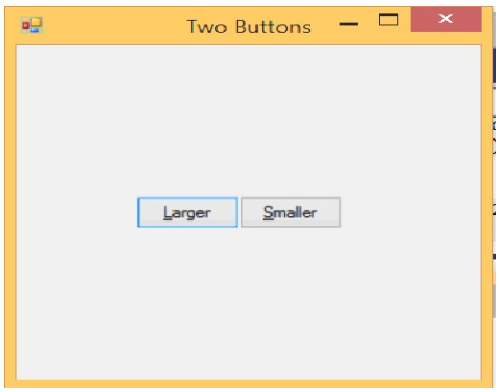


Рис. 4.2. Програмування роботи з кнопками

Лістинг 4.2 (examp02)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp02
{
    public partial class Form1 : Form
    {
        readonly int cxBtn, cyBtn, dxBtn;

        public Form1()
        {
            InitializeComponent();
            Text = "Two Buttons";
            ResizeRedraw = true;
            dxBtn = Font.Height;
            cxBtn = 5 * dxBtn;
            cyBtn = 2 * dxBtn;
            btnLarger.Text = "&Larger";
            btnSmaller.Text = "&Smaller";
            btnLarger.Size = new Size(cxBtn, cyBtn);
            btnSmaller.Size = new Size(cxBtn, cyBtn);
            OnResize(EventArgs.Empty);
        }
        protected override void OnResize(EventArgs e)
        {
            base.OnResize(e);
            btnLarger.Location =
                new Point(ClientSize.Width / 2 - cxBtn - dxBtn / 2,
                (ClientSize.Height - cyBtn) / 2);
            btnSmaller.Location =
                new Point(ClientSize.Width / 2 - dxBtn / 2,
                        (ClientSize.Height - cyBtn) / 2);
        }
        private void btnClick(object sender, EventArgs e)
        {
            Button btn = (Button)sender;
            if (btn == btnLarger)
            {
                Left -=   (int)(0.05 * Width);
                Top -=    (int)(0.05 * Height);
                Width +=  (int)(0.10 * Width);
```

```
                Height += (int)(0.10 * Height);
            }
            else
            {
                Left += (int)(Width/22f);
                Top += (int)(Height/22f);
                Width -= (int)(Width/11f);
                Height -= (int)(Height/11f);
            }
        }
    }
}
```

Конструктор обчислює три значення і зберігає в полях: *cxBtn* і *cyBtn* визначають ширину і висоту кожної кнопки, а *dxBtn* - відстань між двома кнопками. Всі три значення ґрунтуються на властивості *Height* властивості *Font* форми. Оскільки елементи керування успадковують останню властивість від предків, такий самий розмір має шрифт кнопок. Висота кнопок встановлюється вдвічі більше висоти шрифту, а їх ширина – уп’ятеро більша за ширину шрифту. Конструктор встановлює лише розмір кожної кнопки, а чи не їх розташування. Оскільки розташування кнопки залежить від розміру клієнтської області, вона не встановлюється до того часу. доки не викликається метод *OnResize*. У перший раз він викликається в останньому операторі конструктора.

Текст кожної кнопки починається з амперсанта (*&*), що призводить до того, що перша літера підкреслюється. Підкреслена літера працює як клавіша швидкого доступу. Під час роботи програми можна на короткий час натиснути клавішу *Alt* і літеру. При цьому викликається подія *Click*. Перемикатися між кнопками можна, натискаючи або клавішу *Tab*, або *клавіші-стрілки*. Під час перемикання між кнопками переміщається пунктирна рамка всередині кнопки, вказуючи на те, що кнопка має фокус уведення. Кнопка, яка має фокус введення, викликає подію *Click* при натисканні пробілу. У нашому випадку викликається подія *Click* натисканням клавіш *Alt+L* та *Alt+S*.

Форма, на якій розміщуються всі елементи керування, має властивості, що дозволяють призначати кнопку за замовчуванням та кнопку скасування. Кнопка за замовчуванням натискається при будь-якому натисканні клавіш введення незалежно від того, на якому елементі керування форми в цей момент знаходиться фокус. Скажімо, за наявності кнопки за замовчуванням вводиться інформація у *текстбокс*, тоді досить натиснути кнопку *Enter* активувати кнопку, тобто викликати подію *Click*. Винятоком є випадки, коли елемент керування, на якому знаходиться фокус, є іншою кнопкою. Частина призначення кнопки за замовчуванням:

```
private void Form1_Load(object sender, EventArgs e)
{

AcceptButton=button1;// здесь имя кнопки, которую требуется сделать
                     // кнопкой по умолчанию
}
```

У діалогових вікнах зазвичай стандартними кнопками є кнопки з написами *OK*, *Open* (Відкрити) або *Save* (Зберегти). Ця кнопка спрацьовує, коли фокус введення має інший елемент керування та користувач натискає *Enter*. Крім того, кнопка *Cancel* (Скасувати) зазвичай є кнопкою скасування, яка спрацьовує при натисканні клавіші *Esc*.

На початку методу *ButtonOnClick* параметр об'єкта приводиться до типу *Button*. Тепер, порівнявши цей об'єкт з об'єктами *btnLarger* та *btnSmaller*, які конструктор зберіг у полях, метод може ухвалити рішення про подальші дії. Залежно від того, яка кнопка була натиснута, розмір вікна або збільшується або зменшується на 10%. Крім того, вікно зсувається на 5%, так що залишається на тому самому місці екрана, Зміна розмірів вікна призводить до виклику методу *OnResize*, в якому кнопки переміщуються в новий центр клієнтської області.

Приклад 3

У лістингу 4.3 наведено програму керування зовнішнім відображенням кнопки з використанням властивості *FlatStyle*. Ця властивість має чотири значення, наведені вище в описі класу *Button*. Результат роботи програми наведено на рис. 4.3.

Лістинг 4.3 (examp03)

```
using System.Drawing;
using System.Windows.Forms;

namespace exam3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            Text= "Button Styles";
            int y = 0;

            foreach (FlatStyle fs in Enum.GetValues(typeof(FlatStyle)))
            {
                Button btn = new Button();
                btn.Parent = this;
                btn.FlatStyle = fs;
                btn.Text = fs.ToString();
                btn.Location = new Point(50, y += 50);
            }
        }
    }
}
```

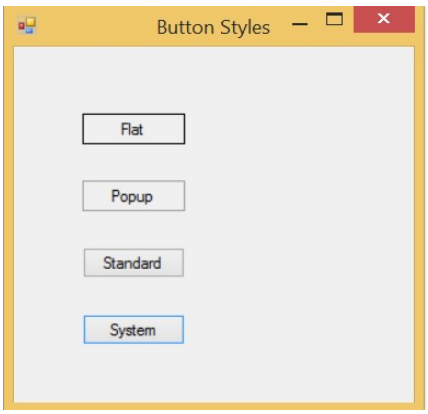


Рис. 4.3. Відображення кнопок

Стиль *Standard* збігається зі стилем *System*, але в цьому екранному знімку кнопка *System* має фокус введення і є стандартною кнопкою.

Приклад 4.

Нижче наведено програму, що створює 29 елементів керування *Button* та реалізує шістнадцятковий калькулятор. Програма *HexCalc* (лістинг 4.4) працює з 64-бітними цілими числами без знака і може виконувати додавання, віднімання, множення, поділ, обчислення залишку, порозрядні *І*, *АБО*, *виключне АБО*, а також порозрядний зсув. Ось як виглядає її вікно:

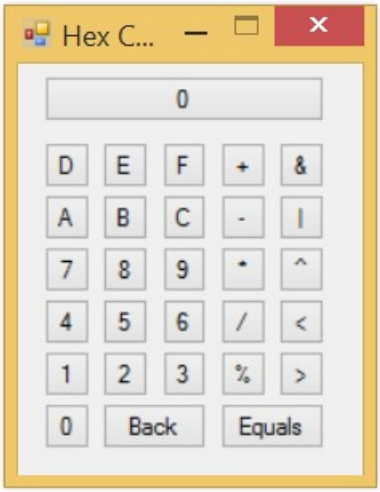


Рис. 4.4. Шіснадцятковий калькулятор

Працювати з *HexCalc* можна за допомогою миші чи клавіатури. Першим йде операнд (він вводиться за допомогою клавіатури або кнопок), потім операція та другий операнд. Щоб переглянути результати, потрібно натиснути кнопку *Equals* або натиснути клавіші *"рівно"* (=) або *Enter*. Змінювати введені дані можна, натиснувши кнопку Back або натиснувши клавішу *Backspace*. Ви можете очистити вікно введення, натиснувши його або натиснувши клавішу *Esc*. У цій програмі не використовується *Forms Designer*. Величезна кількість кнопок з однаковими координатами та розмірами потребує програмного підходу. При цьому створюється клас *CalcButton*, що є підкласом *Button*. Як аргументи конструктора передаються

посилання на батьківський клас, текст, символ натиснутої клавіші, розташування та розмір. Конструктор *HexCalc* містить 29 операторів *new CalcButton*, що створюють усі кнопки. Для кнопок застосовується традиційна система координат діалогового вікна, але при цьому безпосередньо викликається коротка версія *Scale*, яка однаково змінює масштаб у всіх напрямках. Це дозволяє зберегти більшість кнопок квадратними.

Лістинг 4.4 (examp04)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace exam06
{

    // Работа с 16-м калькулятором
    public partial class Form1 : Form
    {
       Button btnResult;
       ulong ulNum = 0;
       ulong ulFirstNum = 0;
       bool bNewNumber = true;
       char chOperation = '=';

        public Form1()
        {
            InitializeComponent();
            Text = "Hex Calc";
            FormBorderStyle = FormBorderStyle.FixedSingle;
            MaximizeBox = false;
            new CalcButton(this,"D",'D',8,24,14,14);
            new CalcButton(this,"A",'A',8,40,14,14);
            new CalcButton(this,"7",'7',8,56,14,14);
            new CalcButton(this,"4",'4',8,72,14,14);
            new CalcButton(this,"1",'1',8,88,14,14);
            new CalcButton(this,"0",'0',8,104,14,14);
            new CalcButton(this,"E",'E',26,24,14,14);
            new CalcButton(this,"B",'B',26,40,14,14);
            new CalcButton(this,"8",'8',26,56,14,14);
            new CalcButton(this,"5",'5',26,72,14,14);
            new CalcButton(this,"2",'2',26,88,14,14);
            new CalcButton(this,"Back",'\x08',26,104,32,14);
            new CalcButton(this,"C",'C',44,40,14,14);
            new CalcButton(this,"F",'F',44,24,14,14);
            new CalcButton(this,"9",'9',44,56,14,14);
            new CalcButton(this,"6",'6',44,72,14,14);
            new CalcButton(this, "3", '3', 44, 88, 14, 14);
            new CalcButton(this, "+", '+', 62, 24, 14, 14);
            new CalcButton(this, "-", '-', 62, 40, 14, 14);
            new CalcButton(this, "*", '*', 62, 56, 14, 14);
            new CalcButton(this,"/",'/',62,72,14,14);
            new CalcButton(this,"%",'%',62,88,14,14);
            new CalcButton(this,"Equals",'=',62,104,32,14);
            new CalcButton(this,"&&",'&',80,24,14,14);
```

```
            new CalcButton(this,"|",'|',80,40,14,14);
            new CalcButton(this,"^",'^',80,56,14,14);
            new CalcButton(this,"<",'<',80,72,14,14);
            new CalcButton(this,">",'>',80,88,14,14);

            btnResult = new CalcButton (this, "0", '\x1B', 8, 4, 86, 14);
            foreach (Button btn in Controls)
                btn.Click += new EventHandler (ButtonOnClick);

            ClientSize =  new Size(102, 126);
            Scale(Font.Height/ 8.0f);
        }
        //Нажатие на клавиши
        protected override void OnKeyPress(KeyPressEventArgs e)
        {
            base.OnKeyPress(e);
            char chKey = Char.ToUpper(e.KeyChar);
            if (chKey == '\x0D')
                chKey = '=';
            CalcButton btn;
            for (int i = 0; i < Controls.Count; i++)
            {
                btn = (CalcButton)Controls[i];
                if (chKey == btn.chKey)
                {
                    InvokeOnClick(btn, EventArgs.Empty);
                    break;
                }
            }
        }
        //Отклик на кнопку
        void ButtonOnClick(object obj, EventArgs ea)
        {
            CalcButton btn = (CalcButton)obj;

            if (btn.chKey =='\x08')
            ulNum/=16;
            else if (btn.chKey == '\x1B')
            ulNum = 0;
            else if (Char.IsLetterOrDigit(btn.chKey))
// Шестнадцатиричная цифра
            {
            if (bNewNumber)
            {
                ulFirstNum = ulNum;
                ulNum = 0;
                bNewNumber = false;
            }
            if (ulNum <= ulong.MaxValue >> 4)
                ulNum = 16 * ulNum + (ulong)
                    (btn.chKey -
                    (Char.IsDigit(btn.chKey) ? '0' : 'A' - 10 ));
        else // Операция
        {
        if (!bNewNumber)
        {
        switch(chOperation)
        {
        case '=':  break;
        case '+': ulNum += ulFirstNum; break;
        case '-': ulNum = ulFirstNum - ulNum; break;
        case '*' : ulNum*= ulFirstNum; break;
```

```
            case '&' : ulNum = ulFirstNum&ulNum; break;
            case '|' : ulNum = ulFirstNum|ulNum; break;
            case '^': ulNum = ulFirstNum ^ ulNum; break;
            case '<': ulNum = ulFirstNum << (int)ulNum; break;
        case '>' : ulNum = ulFirstNum>>(int)ulNum; break;
        case '/' : ulNum = ulNum!=0 ? ulFirstNum/ulNum:
            ulong.MaxValue; break;
        case '%' : ulNum = ulNum!=0 ? ulFirstNum%ulNum:ulong.MaxValue; break;
            default: ulNum = 0; break;
            }
            }
            bNewNumber = true;
            chOperation = btn.chKey;
           }
            btnResult.Text = String.Format("{0:X}",ulNum);
        }
        }
        //Класс конопки калькулятора
        class CalcButton: Button
        {
            public char chKey;
            public CalcButton(Control parent, string str, char chkey,
            int x, int y, int cx, int cy)
            {
             Parent = parent;
             Text = str;
             chKey = chkey;
             Location = new Point(x, y);
             Size = new Size(cx, cy);
             SetStyle(ControlStyles.Selectable, false);
            }
        }
}
```

У *HexCalc* немає потреби, щоб кнопки отримували фокус введення. Штрихова лінія навколо кнопки, що означає, що кнопка має фокус введення, виглядала б не дуже добре. Крім того, переміщувати фокус введення між кнопками за допомогою клавіші *Tab* було б безглуздо. І, нарешті, клавіші, які сприймаються цією програмою, більше ніж кнопоки. Щоб переправити клавіатурні події формі, у кожної кнопки стиль *Selectable* встановлено у *false*, завдяки чому кнопки не отримують фокус введення. Метод *OnKeyPress* проходить по всіх елементах масиву *Controls* і знаходить кнопку відповідну натиснутій клавіші. Потім викликається метод *InvokeOnClick*, імітуючи подію кнопки *Click*. Таким чином, метод *ButtonOnClick* обробляє натискання кнопок та натискання їх клавіатурних еквівалентів.

4.2. Перемикачі та групові блоки (CheckBox, RadioButton, GroupBox)

CheckBox (прапорець) складається з маленького прямокутника та текстового рядка. При натисканні елемента керування (або натискання пробілу, коли елемент має фокус введення) у прямокутнику з’являється позначка. Якщо ще раз натиснути елемент керування, позначка зникає. *CheckBox* успадковано від *ButtonBase* і застосовується для прийняття команди користувача із двома або трьома станами. Якщо властивість

ThreeState встановлено в *true*, то властивість *CheckState* елемента *CheckBox* може приймати одне з наступних трьох значень:

Checked	Елемент *CheckBox* відзначений.
Unchecked	Елемент *CheckBox* не відзначений.
Indeterminate	У цьому стані елемент *CheckBox* недоступний.

Стан *Indeterminate* може бути встановлений лише програмно, а не користувачем. Це зручно, якщо ви хочете повідомити користувача, що опція не була встановлена. Для отримання поточного стану у вигляді булевського значення можна звернутися до *Checked* властивості.

Події *CheckedChanged* і *CheckStateChanged* виникають, коли змінюється властивість *CheckState* або *Checked*. Перехоплення цих подій може стати в нагоді для встановлення інших значень на основі нового стану *CheckBox*. Подія *CheckedChanged* для кількох елементів *CheckBox* обробляється наступним методом:

```
private void checkBoxChanged(object sender, EventArgs e)
{
  CheckBox checkBox = (CheckBox)sender;
  MessageBox.Show("Новое значение " + checkBox.Name + " равно " +
  checkBox.Checked.ToString());
}
```

При зміні стану кожного з цих елементів відображається вікно повідомлення з ім’ям *CheckBox* та його новим станом.

Приклад 5.

У лістингу 4.5. наведено програму створення 4 елементів керування *CheckBox*, що дозволяють задати 4 атрибути шрифту: напівжирний, курсив, підкреслений та перекреслений.

Лістинг 4.5. (exam05)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;


namespace exam05
{
    public partial class Form1 : Form
    {
        string[] strText = {"Bold", "Italic", "Underline", "Strikeout"};
        public Form1()
        {
            InitializeComponent();
            Text = "CheckBox";
            int cyText = 2*Font.Height;
            int cxText = cyText / 2;
```

```
            label1.Text = Text + " Sample Text";
            label1.AutoSize = true;
              for (int i = 0; i < 4; i++)
              {
                CheckBox chkbox = new CheckBox();
                chkbox.Parent = this;
                chkbox.Text = strText[i];
                chkbox. Location = new Point(2 * cxText,
               (4 + 3 * i) * cyText / 2);
               chkbox.Size = new Size(12 * cxText, cyText);
               chkbox.CheckedChanged+= new EventHandler(ChkB);
                }
        }
        void ChkB(object obj, EventArgs ea)
        {
            FontStyle fs = 0;
            FontStyle []afs = {FontStyle.Bold, FontStyle.Italic,
            FontStyle.Underline, FontStyle.Strikeout};
            for (int i = 0; i < 4; i++)
            if( ((CheckBox) Controls[i+1]).Checked)
            fs |= afs[i];
            label1. Font = new Font(label1.Font, fs);
        }
    }
}
```

Змінна *cyText* визначена висотою фона форми. Змінна *cxText* має вдвічі менше значення. Ці змінні дозволяють встановити властивості *Location* і *Size* кожного елемента керування.

В обробнику події *CbeckedChanged* за значеннями властивості *Checked* кожного з чотирьох елементів керування формується значення змінної *FontStyle*. Потім створюється новий об’єкт *Font* та виводиться текст.

Результат роботи програми наведено на рис. 4.5.

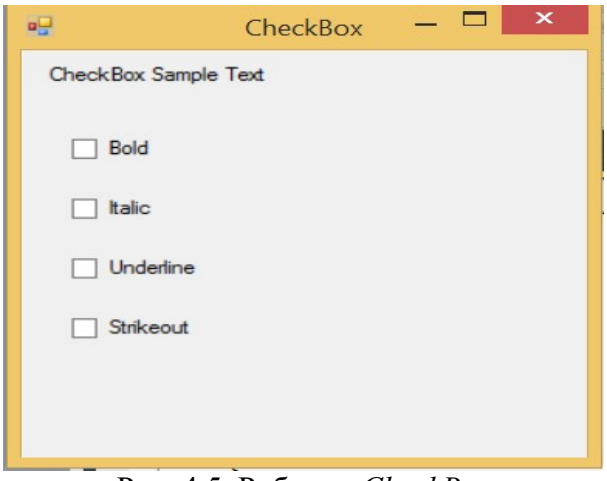


Рис. 4.5. Робота з *CheckBox*

RadioButton (перемикач) успадкований від *ButtonBase*. Перемикачі зазвичай використовуються у складі груп. Іноді, як їх називають, кнопками вибору (*option buttons*) перемикачі дають можливість користувачу вибирати одну з декількох опцій. Коли ви використовуєте безліч елементів керування *RadioButton* в одному контейнері, вибраним може бути лише один із них. Тому якщо у вас є три опції, наприклад, *Red*, *Green* і *Blue*, і якщо вибрано опцію *Red*, а користувач натискає *Blue*, то *Red* автоматично відключається.

Властивість *Appearance* набуває значення з переліку *Appearance*. Вона може бути або *Button*, або *Normal*. Коли вибирається *Normal*, перемикач виглядає як маленький кружок з міткою поруч із ним. Вибір його заповнює кружок, вибір іншого перемикача з тієї ж групи скасовує вибір поточного вибраного перемикача і робить кружок порожнім. При встановленні значення *Appearance* рівним *Button* перемикач виглядає подібно до стандартної кнопки, але працює подібно до перемикача - вибрана кнопка натиснута, не вибрана - відпущена. Властивість *CheckedAlign* визначає, де знаходиться кружок по відношенню до тексту напису. Він може бути над текстом, під ним, праворуч чи ліворуч. Подія *CheckedChanged* виникає щоразу, коли значення властивості *Checked* змінюється. Подібним чином можна виконати інші дії на основі нового значення елемента керування.

Усередині групи кнопок-перемикачів клавіші-стрілки переводять фокус уведення з однієї кнопки на іншу. При зміні фокусу змінюється також і кнопка-перемикач. Клавіша *Tab* дозволяє перейти від групи перемикачів до наступного елемента керування. При переході за допомогою клавіші *Tab* до групи перемикачів цей перемикач отримує фокус введення. Для кожної групи перемикачів створюється елемент керування *GroupBox*, який є дочірнім елементом форми. Усі об’єкти *RadioButton* є дочірніми елементами *GroupBox*. Як зазначалося, у *RadioButton* і *CheckBox* подія *CheckedChanged* виникає щоразу, коли значення властивості *Checked* змінюється.

Приклад 6.

У програмі, представленій у лістингу 4.6., малюється еліпс залежно від стану восьми перемикачів і одного прапорця. На рис. 4.6. показано роботу програми.

Лістинг 4.6. (examp06)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;


namespace exam06
{

    //Пример работы с радиокнопками
    public partial class Form1 : Form
    {
        bool bFillEllipse;
        Color colorEllipse;
        public Form1()
        {
            InitializeComponent();
            Text = "Radio Buttons Demo";
            ResizeRedraw = true;
            string[] astrColor = { "Black", "Blue", "Green", "Cyan",
```

```
        "Red", "Magenta", "Yellow", "White"};
         groupBox1.Text = "Color";
         groupBox1.Size = new Size(9 * Font.Height,
         (int)(1.5 * Font.Height * astrColor.Length));
         RadioButton radiobtn;
         for (int i = 0; i < astrColor.Length; i++)
         {
             radiobtn = new RadioButton();
             radiobtn.Parent = groupBox1;
             radiobtn.Text = astrColor[i];
             radiobtn.Location = new Point(Font.Height,
             3 * (i + 1) * Font.Height / 2);
             radiobtn.Size = new Size(7 * Font.Height,
             3 * Font.Height / 2);
             radiobtn. CheckedChanged +=
             new EventHandler(RadioChanged);
             if (i == 0) radiobtn.Checked = true;
         }
         CheckBox chkbox = new CheckBox();
         chkbox.Parent = this;
         chkbox.Text = "fill Ellipse";
         chkbox.Location = new Point(Font.Height,
         3 * (astrColor.Length + 2) * Font.Height / 2);
         chkbox.Size = new Size(Font.Height * 7, 3 * Font.Height / 2);
         chkbox. CheckedChanged += new EventHandler(CheckBoxChanged);
    }
    void RadioChanged(object obj, EventArgs ea)
    {
         RadioButton rbtn = (RadioButton)obj;
         if (rbtn.Checked)
         {
             colorEllipse = Color.FromName(rbtn.Text);
             Invalidate(false);
         }
    }

  void CheckBoxChanged(object obj, EventArgs ea)
  {
    bFillEllipse = ((CheckBox)obj). Checked;
    Invalidate(false);
  }

  protected override void OnPaint(PaintEventArgs e)
  {
      base.OnPaint(e);
     Graphics gr = e.Graphics;
     Rectangle rect = new Rectangle(groupBox1.Right, 0,
     ClientSize.Width - groupBox1.Right - 1, ClientSize.Height - 1);
     if(bFillEllipse)gr.FillEllipse(new SolidBrush(colorEllipse), rect);
     else gr.DrawEllipse(new Pen(colorEllipse), rect);
  }
 }
```

На початку конструктора визначається масив із 8 назв кольорів. Усі вертикальні координати та розміри у програмі обчислюються так, щоб вони вмістили додаткові назви кольорів із цього масиву. Конструктор створює елемент керування *GroupBox*. Батьківським елементом групового блоку є форма. Потім конструктор створює 8 перемикачів – дочірніх елементів групового блоку. Наприкінці циклу *for* властивість *Checked* встановлюється в

true для першої кнопки. Конструктор завершується створенням елемента керування *CheckBox* - дочірнього елемента форми.

Активація кнопок виконується їх натисканням або під час використання клавіш-стрілок. При цьому генерується подія (відгук), тобто викликається метод *RadioButton.CheckedChanged*. У цьому методі змінюється значення змінної *colorEllipse* об'єкта *Color*. За допомогою *Invalidate()* викликається функція *Paint()* і еліпс оконтурюється вибраним кольором. Аналогічно за допомогою елемента керування *CheckBox* змінюється колір заливки еліпса.

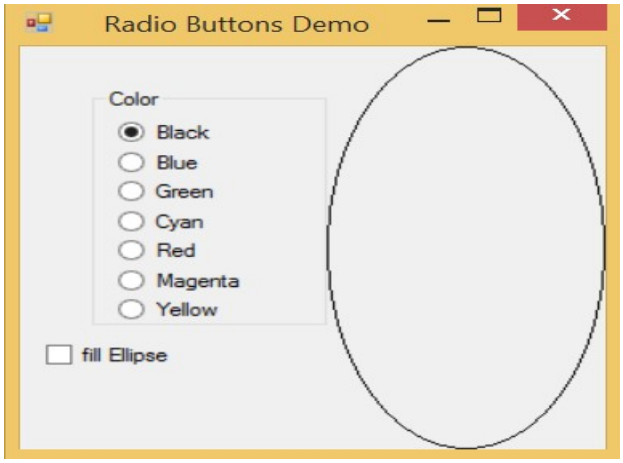


Рис. 4.6. Робота з перемикачами.

4.3. Текстові елементи керування (TextBox, RichTextBox і MaskedTextBox)

У бібліотеці *.NET Framework 2.0* у просторі імен *System.Windows.Forms* є абстрактний клас *TextBoxBase*, який забезпечує похідні від нього класи безліччю можливостей для редагування тексту. Ланцюжок успадкування цих класів виглядає так:

На малюнку видно, що є 5 класів, здатних породжувати текстові елементи керування:

1. TextBox
2. RichTextBox
3. MaskedTextBox
4. DataGridTextBox
5. DataGridViewTextBoxEditingControl

Ми розглянемо перші три класи які є найпростішими. Кожен має властивість *Text* типу *string*, у якій зберігається інформація яка редагується в елементі керування. Більшість інших властивостей набувають булевого значення і визначають режим роботи елемента.

TextBox.

Елемент керування *TextBox* - один з найчастіше використовуваних. *TextBox*, *RichTextBox* та *MaskedTextBox* успадковані від *TextBoxBase*. Клас *TextBoxBase* представляє такі властивості, як *MultiLine* та *Lines*. Властивість *MultiLine* - булівське значення, що дозволяє елементу керування *TextBox* відображати текст у більш ніж одному рядку. При цьому кожен рядок у текстовому вікні є частиною масиву рядків. Цей масив доступний через властивість *Lines*. Властивість *Text* повертає повний вміст текстового вікна у вигляді рядка. *TextLength* - загальна довжина тексту. Властивість *MaxLength* обмежує довжину тексту певною величиною. *SelectedText*, *SelectionLength* та *SelectionStart* мають справу з поточним виділеним текстом у текстовому вікні. Виділений текст підсвічується, коли елемент керування отримує фокус. *TextBox* додає багато цікавих властивостей. *AcceptsReturn* - булівське значення, що дозволяє *TextBox* сприймати клавішу <*Enter*> як символ нового рядка чи активізувати кнопку за замовчуванням на формі. Коли ця властивість має значення *true*, то натискання клавіші <*Enter*> створює новий рядок у *TextBox*. Властивість *CharacterCasing* визначає регістр тексту у вікні. Перелік *CharacterCasing* містить три значення: *Lower*, *Normal* та *Upper*. Значення *Lower* переводить у нижній регістр весь текст, незалежно від того, як його було введено, *Upper* переводить весь текст у верхній регістр, а *Normal* відображає текст так, як його було введено.

Властивість *PasswordChar* елемента керування *TextBox* дозволяє вказати символ, який відображатиметься під час введення користувачем усіх символів у текстовому вікні. Це застосовується під час введення паролів та *PIN-кодів*. Властивість *Text* поверне дійсний текст; властивість *PasswordChar* стосується лише відображення символів.

Приклад 7.

Приклад введення пароля у лістингу 4.7, який докладно відкоментовано. На рис. 4.7. наведено варіант роботи цієї програми.

Лістинг 4.7. (examp07)

```
using System;
// Другие директивы using удалены, поскольку они не используются в данной
программе
using System.Drawing;
using System.Windows.Forms;
// Программа для ввода пароля в текстовое поле, причем при вводе вместо
вводимых
// символов некто, "находящийся за спиной пользователя", увидит только
звездочки
namespace examp07
{
```

```
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            base.Text = "Введи пароль";
            textBox1.Text = null;
            textBox1.TabIndex = 0;
            textBox1.PasswordChar = '*';
            textBox1.Font = new Font("Courier New", 9.0F);
            // или textBox1.Font = new Font(FontFamily.GenericMonospace, 9.0F);
            label1.Text = "";
            label1.Font = new Font("Courier New", 9.0F);
            button1.Text = "Покажи паспорт";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Обработка события "щелчок на кнопке"
            label1.Text = textBox1.Text;
        }
    }
}
```

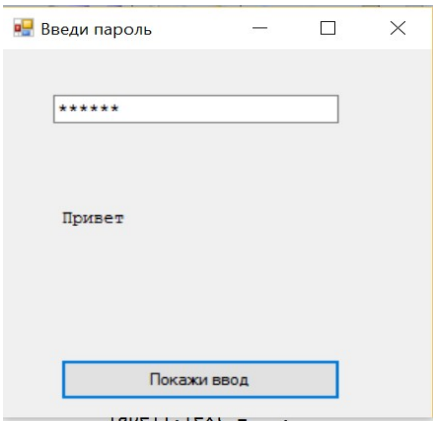


Рис. 4.7. Введення пароля

Важливим є контроль даних, що вводяться. Одним із способів перевірки введення числа є використання методу *TryParse*.

Приклад 8

Наведемо типову програму, яка вводить через текстове поле число, при натисканні кнопки витягує із нього квадратний корінь і виводить результат на мітку. У разі введення не числа повідомляє користувача про це. Текст програми наведено на лістингу 4.8. Цей лістинг досить повно прокоментовано.

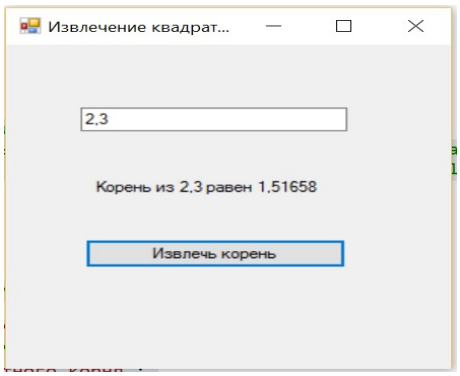
Лістинг 4.8. (examp08)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

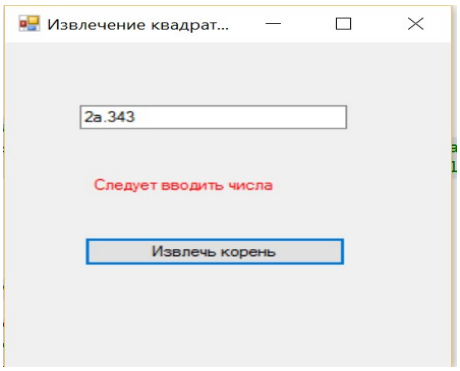
```
    // Программа вводит через текстовое поле число, при щелчке на командной кнопке
    // извлекает из него квадратный корень и выводит результат на метку label1. В
случае
    // ввода не числа сообщает пользователю об этом красным цветом также на метку
label1
    namespace examp08
    {
        public partial class Form1 : Form
        {
            public Form1()
            { // Инициализация компонентов формы
                InitializeComponent();
                button1.Text = "Извлечь корень";
                label1.Text = null; // или = ""
                this.Text = "Извлечение квадратного корня";
                textBox1.Clear(); // Очистка текстового поля
                textBox1.TabIndex = 0; // Установка фокуса в текстовом поле
            }
            private void button1_Click(object sender, EventArgs e)
            { // Обработка события щелчок на кнопке Извлечь корень
                Single X; // - из этого числа будем извлекать корень
                // Преобразование из строковой переменной в Single:
                bool Число_ли = Single.TryParse(textBox1.Text,
                   System.Globalization.NumberStyles.Number,
                   System.Globalization.NumberFormatInfo.CurrentInfo, out X);
                // второй параметр - это разрешенный стиль числа (Integer,
                // шестнадцатиричное число, экспоненциальный вид числа и прочее)
                // третий параметр - форматирует значения на основе текущего языка
                // и региональных параметров из Панели управления - Язык и
региональные
                // стандарты число допустимого формата,
                // метод возвращает значение в переменную X
                if (Число_ли == false)
                { // Если пользователь ввел не число:
                    label1.Text = "Следует вводить числа";
                    label1.ForeColor = Color.Red; // - красный цвет текста на метке
                    return; // - выход из процедуры
                }
                Single Y = (Single)Math.Sqrt(X); // - извлечение корня
                label1.ForeColor = Color.Black; // - черный цвет текста на метке
                label1.Text = string.Format("Корень из {0} равен {1:F5}", X, Y);
            }
        }
    }
```

Тут при натисканні кнопки "Извлечь корень" виконується перевірка за допомогою методу *TryParse*. У цьому методі першим параметром є аналізоване поле *textBox1.text*, другий параметр - стиль числа, що дозволяється ввести. Він може бути цілим, шістнадцятковим, представленим в експоненційному вигляді та інше. Третій параметр показує, на якій основі формується допустимий формат, у нашому випадку ми використовували *CurentInfo*, тобто на основі поточної мови та раціональних параметрів. За замовчуванням при інсталяції українізованої, або русифікованої версії *Windows* роздільником цілої та дробової частини числа є кома. Однак цю установку можна змінити, якщо в *Панелі керування* вибрати значок *Мова* та регіональні стандарти, потім на вкладці *Регіональні параметри* натиснути по кнопці *Налаштування* і на новій вкладці вказати як роздільник крапку

замість коми. В обох випадках *TryParse* враховує будь-який роздільник. Четвертий параметр *TryParse* повертає булеву змінну *true* або *false*, яка повідомляє чи успішно виконано перетворення. На рис. 4.8а та рис. 4.8б показано фрагменти роботи програми. У першому випадку вводиться число. У другому випадку введено літеру та дано повідомлення про помилкове введення.



а)



б)

Рис. 4.8. Обчислення квадратного кореня.

Наведений вище приклад на основі методу *TryParse* дозволяє контролювати повне введення числа. Далі ми покажемо, як можна зовсім по-іншому вирішити задачу контролю даних, що вводяться користувачем. Можна взагалі не давати можливість користувачам вводити нечислові дані.

Приклад 9.

Розглянемо програму на лістингу 4.9., що дозволяє вводити у текстове поле лише цифрові символів з урахуванням встановленого роздільника (точки чи коми).

Лістинг 4.9. (examp09)

```
// Программа разрешает ввод в текстовое поле только цифровых символов, а также
// разделитель целой и дробной части числа
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
программе

namespace examp09
{
    public partial class Form1 : Form
    {
        // Разделитель целой и дробной части числа может быть
        // точкой "." или запятой "," в зависимости от
        // установок в пункте Язык и региональные стандарты
        // Панели управления ОС Windows:
        System.Globalization.CultureInfo Культ = System.Globalization.
                                            CultureInfo.CurrentCulture;
        string ТчкИлиЗпт;
        string nach_zn;

        public Form1()
        {
            InitializeComponent();
            this.Text = "Введите число";
```

```
            // Выясняем, что установлено на данном ПК в качестве
            // разделителя целой и дробной части: точка или запятая:
            ТчкИлиЗпт = Культ.NumberFormat.NumberDecimalSeparator;
        }
        private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
        {
            bool ТчкИлиЗптНАЙДЕНА = false;
            // Разрешаю ввод десятичных цифр:
            if (e.KeyChar == '+' ) return;
            if (e.KeyChar == '-' ) return;
            if (char.IsDigit(e.KeyChar) == true) return;
            // Разрешаю ввод <BackSpace>:
            if (e.KeyChar == (char)Keys.Back) return;
            // Поиск ТчкИлиЗпт в textBox, если IndexOf() == -1, то не найдена:
            if (textBox1.Text.IndexOf(ТчкИлиЗпт) != -1) ТчкИлиЗптНАЙДЕНА =
true;
            // Если ТчкИлиЗпт уже есть в textBox, то запрещаем вводить и ее,
            // и любые другие символы:
            if (ТчкИлиЗптНАЙДЕНА == true) { e.Handled = true; return; }
            // Если ТчкИлиЗпт еще нет в textBox, то разрешаем ее ввод:
            if (e.KeyChar.ToString() == ТчкИлиЗпт) return;
            // В других случаях запрет на ввод:
             e.Handled = true;
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {
            string per = ((TextBox)sender).Text;
            Double voz;
            bool v = Double.TryParse(per, out voz);
            Text = voz.ToString();
            if (per.Length == 1 && char.Parse(per) == '+') return;
            if (per.Length == 1 && char.Parse(per) == '-') return;
            if (per.Length == 0) return;
            if (v == true) {nach_zn = per; return;}
            if (v == false)
                ((TextBox)sender).Text = nach_zn;
            textBox1.Select(per.Length, 0);
        }
    }
}
```

На рис. 4.9. показано фрагмент роботи программи.

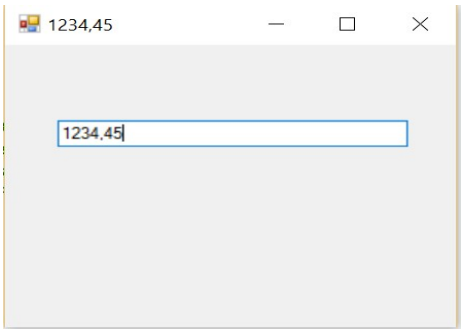


Рис. 4.9. Посимвольний контроль введення числа

Слід зазначити, що контроль введення даних ефективніший з використанням елемента керування *ErrorProvider*, який розглядається нижче.

Крім того, контроль введення даних у *TextBox* може виконуватися за допомогою регулярних виразів. Розглянемо дві задачі. У першому завданні введений рядок перевіряється на відповідність його прізвища російською мовою. У другому завданні перевіряється рядок на введення додатного раціонального числа.

Приклад 10

Суть першого завдання полягає в тому, що після введення програма повинна порівнювати цей рядок з деяким шаблоном і зробити висновок, чи відповідає введене значення шаблону російського прізвища. Текс програми наведено у лістингу 4.10.

Лістинг 4.10. (examp10)

```
// Проверка данных, вводимых пользователем, на достоверность.
// Программа осуществляют синтаксический разбор введенной пользователем
// текстовой строки на соответствие ее фамилии на русском языке
using System.Text;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
программе
 namespace examp10
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
                label1.Text = "Введите фамилию на русском языке:";
                button1.Text = "Проверка";
        }
                private void button1_Click(object sender, System.EventArgs e)
        {
                textBox1.Text = textBox1.Text.Trim();
                if (System.Text.RegularExpressions.Regex.Match(
                textBox1.Text, "^[А-ИК-ЩЭ-Я][а-яА-Я]*$").Success != true)
                MessageBox.Show ("Неверный ввод фамилии", "Ошибка");
        }
    }
}
```

Ключовим моментом програми є перевірка відповідності введеного користувачем текстового рядка та шаблону за допомогою функції *Regex.Match* (від англійської match - відповідати):

```
Regex.Match(textBox1.Text, "^[А-ИК-ЩЭ-Я][а-яА-Я]*$")
```

Символи ^ і $ відповідають початку та кінцю рядка відповідно. Перша літера має бути великою від **А** до **И**, від **К** до **Щ** і **Э** до **Я**, тобто. не допустимим буде введення букв **Й**, **Ъ**, **Ы**, **Ь** як першої літери прізвища. Далі в наступних квадратних дужках вказано діапазон літер або нижнього або верхнього регістрів, причому символ * означає, що другий діапазон символів може зустрітися в рядку нуль або більше разів. Фрагмент програми представлено на рис. 4.10.

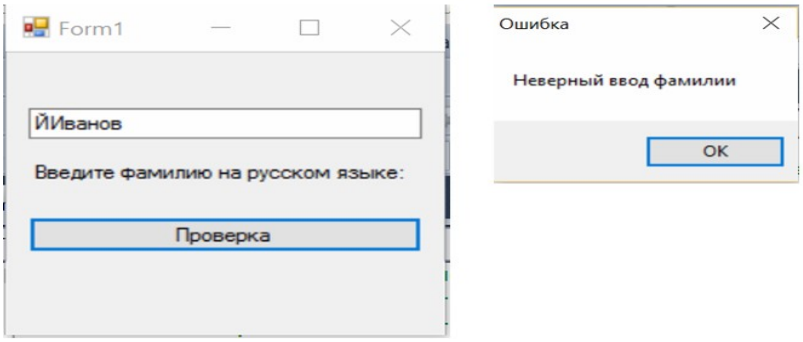


Рис. 4.10. Перевірка коректності введення прізвища російською мовою.

Приклад 11.

У цьому прикладі ми розглянемо друге завдання - перевірку правильності введення додатного раціонального числа. Програмний код показано у лістингу 4.11. Фрагмент програми представлено на рис. 4.11.

Лістинг 4.11. (examp11)

```
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
программе
namespace examp11
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            label1.Text = "Введите положительное рациональное число:";
            button1.Text = "Проверка";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            textBox1.Text = textBox1.Text.Trim();
            if (System.Text.RegularExpressions.Regex.Match(textBox1.Text,
                    "^(([0-9]+.[0-9]*)|([0-9]*.[0-9]+)|([0-9]+))$"
                                                    ).Success == false)
                MessageBox.Show("Некорректный ввод", "Ошибка");
        }
    }
}
```

Цю програму побудовано аналогічно попередній, але наступний шаблон потребує коментарів:

```
"^(([0-9]+.[0-9]*)|([0-9]*.[0-9]+)|([0-9]+))$"
```

Тут символом | між круглими дужками представлено три групи виразів. Перша група дозволяє вводити цифрові символи від 0 до 9 до десяткової точки. Знак плюс (+) означає, що цифровий символ може зустрітися один або більше разів. Таким чином, перша група допускає введення, наприклад, раціонального числа *6*. Аналогічно працює друга група виразів, вона допускає введення, наприклад, числа *.77*. Третя група перевіряє відповідність із будь-якими цілими числами.

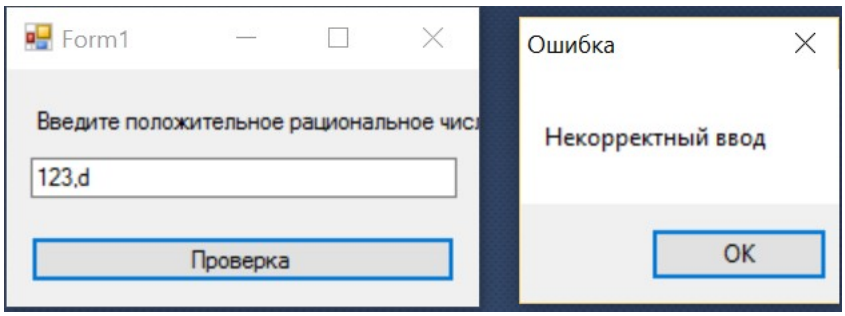


Рис. 4.11. Введення раціонального числа

RichTextBox.

RichTextBox - елемент керування, який служить для редагування тексту з розширеними можливостями форматування. Як можна зрозуміти з назви, *RichTextBox* використовує *Rich Text Format (RTF)* для обробки спеціального форматування. Зміна формату забезпечується властивостями *SelectionFont*, *SelectionColor* і *SelectionBullet*, а форматування параграфів - властивостями *SelectionIndent*, *SelectionRightIndent* та *SelectionHangingIndent*. Усі властивості їх групи *Selection* працюють однаково. Якщо виділено частину тексту, то зміна властивості стосується цього виділеного фрагмента. Якщо виділеного фрагмента немає, то зміни торкаються будь-якого тексту, що вставляється праворуч від поточної позиції вставки. Текст цього елемента керування може бути вилучений з властивості *Text* чи *Rtf*. Властивість *Text* повертає простий текст елемента керування, тоді як *RTF* - форматований текст. Метод *LoadFile* може завантажувати текст із файлу двома різними способами. Він може використовувати або рядок, що представляє шляхове ім’я файлу або потоковий об’єкт. Також можна специфікувати *RichTextBoxStreamType*. У табл. 4.1 перераховано можливі значення *RichTextBoxStreamType*.

Таблиця 4.1

Значення RichTextBoxStreamType

Значення	Опис
PlainText	Інформація, яка пов’язана з форматуванням, відсутня. У тих місцях, де знаходяться об’єкти *OLE*, використовуються пробіли.
RichNoOleObjs	Форматування *Rich Text Format*, але на місці *OLE-об’єктів* знаходяться пробіли.
RichText	Форматований текст *RTF* та *OLE-об’єктами* на місці.
TextTextOleObjs	Простий текст із текстом, який замінює *OLE-об’єкти*.
UnicodePlainText	Те саме, що і *PlainText*, але з кодуванням *Unicode*.

Метод *SaveFile* працює з кількома параметрами та зберігає текст з елемента керування у вказаному файлі. Якщо файл з такою назвою вже існує, він перезаписується.

Приклад 12.

Напишемо дуже простий *RTF-редактор*, який читає як *RTF-файли*, так і звичайні текстові файли в кодуванні *Windows 1251*, але зберігає файли на диск у будь-якому випадку у форматі *RTF*. З цією метою перенесемо з панелі інструментів керування *ToolBox* елементи керування *RichTextBox*, *MenuStrip*, *SaveFileDialog* та *OpenFileDialog*. У спадаючому меню *Файл* передбачено пункти меню, як показано на рис. 4.12. Текст програми наведено у лістингу 4.12.

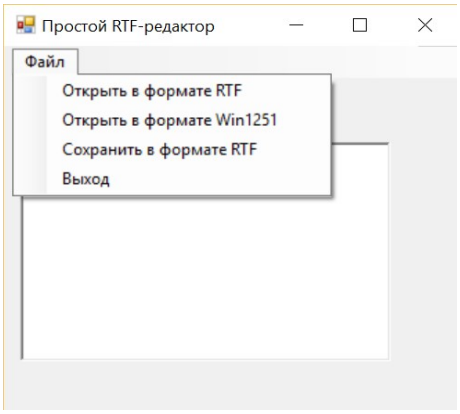


Рис. 4.12. Простий RTF-редактор

Лістинг 4.12. (examp12)

```
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
программе
namespace examp12( Зиборов)
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            base.Text = "Простой RTF-редактор";
            richTextBox1.Clear();
            //openFileDialog1.FileName = @"c:\Text4.rtf";
            saveFileDialog1.Filter = "Файлы RTF (*.RTF)|*.RTF";
            // Подписка на обработку двух событий одной процедурой:
            this.открытьВФориатеRTFToolStripMenuItem.Click += new System.
EventHandler(this.ОТКРЫТЬ);
            this.открытьВФорматеWin1251ToolStripMenuItem.Click += new System.
EventHandler(this.ОТКРЫТЬ);
        }
        private void ОТКРЫТЬ(object sender, EventArgs e)
        {// Процедура обработки событий открытия файла в двух разных форматах:
            // Выясняем, в каком формате открыть файл:
            ToolStripMenuItem t = (ToolStripMenuItem)sender;
            string format = t.Text; // - читаем надпись на пункте меню
            try
            {// Открыть в каком-либо формате:
                if (format == "Открыть в формате RTF")
                {
                    openFileDialog1.Filter = "Файлы RTF (*.RTF)|*.RTF";
```

```
            openFileDialog1.ShowDialog();
            if (openFileDialog1.FileName == null) return;
            richTextBox1.LoadFile(openFileDialog1.FileName);
        }
        if (format == "Открыть в формате Win1251")
        {
            openFileDialog1.Filter = "Текстовые файлы (*.txt)|*.txt";
            openFileDialog1.ShowDialog();
            if (openFileDialog1.FileName == null) return;
            richTextBox1.LoadFile(openFileDialog1.FileName,
            RichTextBoxStreamType.PlainText);
        }
        richTextBox1.Modified = false;
    }
    catch (System.IO.FileNotFoundException Exc)
    {
        MessageBox.Show(Exc.Message + "\nНет такого файла", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
    catch (Exception Exc)
    {
        // Отчет о других ошибках
        MessageBox.Show(Exc.Message, "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}
private void сохранитьВФорматеRTFToolStripMenuItem_Click(
object sender, EventArgs e)
{
    saveFileDialog1.FileName = openFileDialog1.FileName;
    if (saveFileDialog1.ShowDialog() == DialogResult.OK) Запись();
}
void Запись() {
    try
        MessageBox.Show("Запись");
        richTextBox1.SaveFile(saveFileDialog1.FileName);
        richTextBox1.Modified = false;
    }
    catch (Exception Exc)
    {   // Отчет о всех возможных ошибках:
        MessageBox.Show(Exc.Message, "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}
private void выходToolStripMenuItem_Click(object sender, EventArgs e) {
    this.Close();
}
private void Form1_FormClosing(object sender, FormClosingEventArgs e) {
    if (richTextBox1.Modified == false) return;
    // Если текст модифицирован, то выясняем, записывать ли файл?
    DialogResult MBox = MessageBox.Show(
                    "Текст был изменен. \nСохранить изменения?",
                    "Простой редактор", MessageBoxButtons.YesNoCancel,
                                        MessageBoxIcon.Exclamation);
    // YES – диалог;  NO – выход;  CANCEL - редактирование
    if (MBox == DialogResult.No) return;
    if (MBox == DialogResult.Cancel) e.Cancel = true;
    if (MBox == DialogResult.Yes) {
        if (saveFileDialog1.ShowDialog() == DialogResult.OK)
```

```
                { Запись(); return; }
                else e.Cancel = true; // Передумал выходить из ПГМ
            } // - DialogResult.Yes
        }
    }
}
```

В основі процедури *Запись()* також лежить блок *try...catch*: виконати спробу (*try*) збереження файлу (*SaveFile*) і перехопити (*catch*) можливі помилки.

Для закриття програми слід обережно скористатися методом *Exit* об'єкта *Application*, який готує додаток до закриття. Метод *Application.Exit()* не викликає події у формі *Closing*. Можна простежити за поведінкою програми після команди *Application.Exit()* за допомогою відлагоджувача (debug, клавіша <*F11*>). Тут після команди *Application.Exit()* керування перейде до наступного оператора і так до кінця процедури. Аналогічно поводиться метод *Close()* елемента *Form*. При цьому викликається *this.Close*, що викликає подію форми *Closing*. Цей метод закриває форму та звільняє всі ресурси. Таким чином, для негайного виходу з процедури слід комбінувати перераховані методи з *return*.

MaskedTextBox

MaskedTextBox надає можливість обмежити те, що користувач може ввести, а також дозволяє автоматично відформатувати введені дані. Декілька властивостей використовуються для перевірки допустимості формату користувальницького введення. *Mask* - властивість, що містить рядок маски. Маскований рядок аналогічний до рядка форматування. Допустима кількість символів, тип допустимих символів, формат даних - все це встановлюється рядком *Mask*. Елементи строкової властивості *Mask* наведено в таблиці 4.2.

Таблиця 4.2.

Елементи властивості Mask

Маскований елемент	**Пояснення**
0	Очікує будь-яку цифру від 0 до 9
9	Очікується цифра або пробіл
	Очікуються цифра, пробіл, знаки + або -
L	Очікуються ASCII-символи від **a** до **z** або від **A** до **Z**
?	Очікуються будь-які ASCII-символи
&	Очікується будь-який символ. Якщо властивість AsciiOnly дорівнює true, вона спрацьовує як елемент "L"
C	Очікується будь-який некерований символ. Якщо властивість AsciiOnly дорівнює true, то це працює як елемент "?"
A	Очікуються будь-які алфавітно-цифрові символи. Якщо AsciiOnly дорівнює true, то це працює як елемент "L"

Клас, заснований на *MaskedTextProvider*, також може надавати необхідну інформацію для форматування та перевірки. *MaskedTextProvider* можна встановлювати лише передаючи його одному з конструкторів. Три різні властивості повертають текст *MaskedTextProvider*. Властивість *Text* повертає текст, що міститься в елементі керування на даний момент. Воно може відрізнятися залежно від того, чи має елемент фокус, і значення властивості *HidePromptOnLeave*. Запрошення *(prompt)* - це рядок, який бачить користувач і який підказує йому, що потрібно ввести. Властивість *InputText* завжди повертає лише той текст, який було введено користувачем. Властивість *OutputText* повертає текст, сформатований з урахуванням властивостей *IncludeLiterals* і *IncludePrompt*. Якщо, наприклад, маска призначена для введення номера телефону, то рядок *Mask*, мабуть, повинен включати дужки та кілька тире. Це можуть бути літеральні символи, які включаються до властивості *OutputText*, якщо властивості *IncludeLiteral* було привласнено значення *true*. В елементі керування *MaskedTextBox* також є пара додаткових подій. *OutputTextChanged* та *InputTextChanged* виникають тоді, коли змінюються значення *InputText* або *OutputText*.

Приклад 13.

Наведемо приклад контролю за введенням телефонного номера. Програмний код наведено у лістингу 4.13. Тут переноситься з панелі інструментів елемент керування *MaskedTextBox* і в конструкторі форми створюється відгук при завантаженні форми (функція *Form1_Load*). Результат роботи фрагмента програми представлено на рис. 4.13.

Лістинг 4.13. (examp13)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp13
{
    // Класс приложения
    partial class Form1 : Form
    {
        // Объявляем ссылку на элемент поля ввода телефона
        public Form1()    // Конструктор
        {
            InitializeComponent();
        }
        void Form1_Load(object sender, EventArgs e) {
            // Текст заголовка окна
            this.Text = "Элемент MaskedTextBox";
            // Отключаем размеры
            this.MaximizeBox = false;
            this.FormBorderStyle = FormBorderStyle.FixedSingle;
            // Создаем элемент маскированного поля ввода телефона
```

```
                //maskedTextBox1 = new MaskedTextBox();
                maskedTextBox1.Parent = this;// Привязываем к форме
                // Настраиваем маскированное поле ввода
                int x = (this.ClientSize.Width - maskedTextBox1.Width) / 2;
                int y = this.ClientSize.Height / 3 - maskedTextBox1.Height;
                maskedTextBox1.Location = new Point(x, y);
                maskedTextBox1.Mask = "00-00-00";
                Font font = new Font("Arial", 12, FontStyle.Bold);
                maskedTextBox1.Font = font;
                maskedTextBox1.ForeColor = Color.Blue;// Синий
                // Регистрируем событие, когда символ не может быть принят полем
                maskedTextBox1.MaskInputRejected +=
maskedTextBox1_MaskInputRejected;
                // Создаем и настраиваем текстовую метку
                Label label = new Label();
                label.Text = "Введите номер телефона";
                label.Parent = this;
                label.Width = this.ClientSize.Width;
                label.Font = font;
                label.ForeColor = Color.FromArgb(255, 0, 0);// Красный
                // Позиционируем метку над полем ввода
                label.Location = new Point(10, maskedTextBox1.Location.Y -
                label.Height - 10);
            }
     // Содаем объект всплывающей подсказки
            ToolTip toolTip = new ToolTip();

          void maskedTextBox1_MaskInputRejected(object sender,
MaskInputRejectedEventArgs e)
            {
                if (maskedTextBox1.MaskFull) {
                    toolTip.ToolTipTitle = "Ввод отклонен - поле заполнено";
                    // Позиционируем всплывающую подсказку относительно поля ввода
                    toolTip.Show("Вы пытаетесь ввести \n слишком длинный номер",
                      maskedTextBox1, // Привязка к элементу
                    -maskedTextBox1.Location.X,// Сдвинули в начало окна
                      maskedTextBox1.Height, // Сдвинули к низу поля ввода
                    3000);// Подсказка ждет 3 секунды
                }
                else if (e.Position >= maskedTextBox1.Mask.Length) {
                    toolTip.ToolTipTitle = "Ввод отклонен - выход за границу";
                    //Позиционируем всплывающую подсказку относительно поля ввода
                    toolTip.Show
                    ("Не могу добавить новый символ\nза границей поля ввода",
                      maskedTextBox1, // Привязка к элементу
                    -maskedTextBox1.Location.X,// Сдвинули в начало окна
                      maskedTextBox1.Height, // Сдвинули к низу поля ввода
                    3000);// Подсказка ждет 3 секунды
                }
                else {
                    toolTip.ToolTipTitle = "Ввод отклонен - нужны цифры";
                    // Позиционируем всплывающую подсказку относительно поля ввода
                    toolTip.Show(
                        "Номер телефона должен состоять только из цифр (0-9)",
                          maskedTextBox1, // Привязка к элементу
                        -maskedTextBox1.Location.X,// Сдвинули в начало окна
                          maskedTextBox1.Height, // Сдвинули к низу поля ввода
                        3000);// Подсказка ждет 3 секунды
                }
            }
        }
    }
```

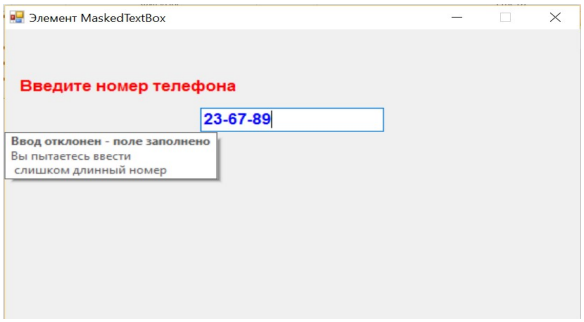


Рис. 4.13. Контроль введення телефонного номера.

4.4. Робота зі списками (ListBox, CheckedListBox і ComboBox).

ComboBox, *ListBox* та *CheckedListBox* – усі вони успадковані від класу *ListControl*. Цей клас визначає деяку базову функціональність керування списками. Найголовніше у використанні спискових елементів керування - це додавання та вибір елементів списку. Те, який список потрібно застосовувати, в основному визначається тим, як його передбачається використовувати, та типом даних, які повинні міститися в ньому. Якщо необхідно мати можливість множинного вибору або користувачеві потрібно бачити в будь-який момент кілька позицій списку, то найкраще підійдуть *ListBox* або *CheckListBox*. Якщо ж у списку може бути обрано лише один елемент, то в цьому разі більше підійде *ComboBox*.

ListBox

Елемент *ListBox* є списком. Ключовою властивістю цього елемента є властивість *Items*, яка й зберігає набір всіх елементів списку.

Особливістю *ListBox* є те, що елементом списку може бути рядок чи об’єкт. У першому випадку *ListBox* назвемо "простим". У другому випадку - *ListBox* з об’єктами.

Для простого *ListBox* елементи до списку можуть додаватися як під час розробки, так і програмним способом. У *Visual Studio* у вікні *Properties* (Властивості) елемента *ListBox* ми можемо знайти властивість *Items*. Після подвійного натискання по властивість нам відобразиться вікно (рис. 4.14) для додавання елементів до списку:

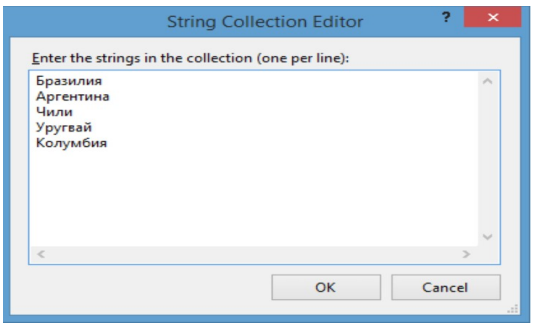


Рис. 4.14. До роботи з *ListBox*.

У порожньому полі ми вводимо по одному елементу списку - по одному на кожному рядку. Після цього всі додані нами елементи з’являться у списку, і ми зможемо ними керувати:

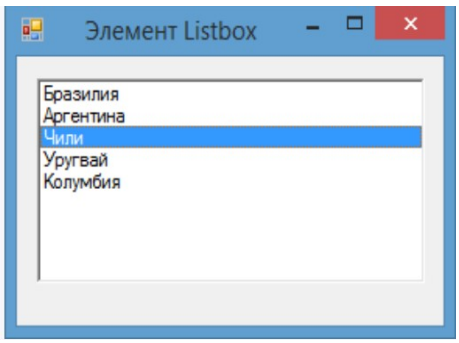


Рис. 4.15. До роботи з *ListBox*

Для додавання нового елемента до цієї колекції, а значить і до списку, треба використовувати метод *Add*, наприклад: *listBox1.Items.Add* (Новий елемент). При використанні цього методу кожен елемент додається в кінець списку.

Можна додати кілька елементів, наприклад, масив рядків. Для цього використовується метод *AddRange*:

```
string[] countries =
{ "Бразилия", "Аргентина", "Чили", "Уругвай", "Колумбия" };
listBox1.Items.AddRange(countries);
```

Вставка виконується за певним індексом списку за допомогою методу Insert:

```
listBox1.Items.Insert(1, "Парагвай");
```

В даному випадку ми вставляємо елемент на другу позицію у списку, тому що відлік позицій починається з нуля.

Для видалення елемента за його текстом використаємо метод *Remove*:

```
listBox1.Items.Remove("Чили");
```

Щоб видалити елемент за його індексом у списку, використовується метод *RemoveAt*:

```
listBox1.Items.RemoveAt(1);
```

Крім того, можна очистити відразу весь список, за допомогою *Clear*:

```
listBox1.Items.Clear();
```

Елемент у списку можна знайти за індексом. Наприклад, отримаємо 1 елемент списку:

```
string firstElement = listBox1.Items[0];
```

Метод *Count* дозволяє визначити кількість елементів у списку:

```
int number = listBox1.Items.Count();
```

При виділенні елементів списку ними можна керувати як через індекс, так і через обраний елемент. Отримати обрані елементи можна за допомогою таких властивостей елемента *ListBox*:

- *SelectedIndex*: повертає або встановлює номер виділеного списку. Якщо виділені елементи відсутні, тоді властивість має значення -1
- *SelectedIndices*: повертає або встановлює колекцію виділених елементів у вигляді набору їх індексів
- *SelectedItem*: повертає або встановлює текст виділеного елемента
- *SelectedItems*: повертає або встановлює виділені елементи у вигляді колекції

За замовчуванням список підтримує виділення одного елемента. Щоб додати можливість виділення кількох елементів, необхідно встановити у його властивості *SelectionMode* значення *MultiSimple*.

Щоб виділити елемент програмно, треба застосувати метод *SetSelected*(int index, bool value), де *index* – номер виділеного елемента. Якщо другий параметр - *value* має значення *true*, то елемент за вказаним індексом виділяється, якщо *false*, то виділення навпаки приховується:

```
listBox1.SetSelected(2, true);
```

Для зняття виділення з усіх виділених елементів використовується метод *ClearSelected*.

З усіх подій елемента *ListBox* треба відзначити в першу чергу подію *SelectedIndexChanged*, яка виникає при зміні виділеного елемента:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        string[] countries = { "Бразилия", "Аргентина", "Чили",
        "Уругвай", "Колумбия" };
        listBox1.Items.AddRange(countries);

        listBox1.SelectedIndexChanged +=
listBox1_SelectedIndexChanged;
    }
   void listBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        string selectedCountry =
listBox1.SelectedItem.ToString();
        MessageBox.Show(selectedCountry);
    }
}
```

У цьому випадку при виборі елемента списку відображатиметься повідомлення з виділеним елементом.

Розглянемо використання простого *ListBox*, тобто. використання списку рядків.

Приклад 14

Написати програму, яка змінює колір фону форми *BackColor*, перебираючи константи кольору за допомогою *ListBox*. У лістингу 4.14. наводиться текст програми, яка повністю відкоментована. Фрагмент роботи програми наведено на рис. 4.16.

Лістинг 4.14. (examp14)

```
// Программа меняет цвет фона формы BackColor, перебирая константы цвета,
// предусмотренные в Visual Studio 2019, с помощью элемента управления ListBox
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
программе
namespace examp14
{
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
            // Получаем массив строк имен цветов из перечисления KnownColor
        //Enum.GetNames- Возвращает массив имен констант в указанном
перечислении.
           // string[] ВсеЦвета = Enum.GetNames(typeof(KnownColor));
            string[] ВсеЦвета = Enum.GetNames(typeof(KnownColor));
            listBox1.Items.Clear();
            // Добавляем имена всех цветов в список listBox1:
            listBox1.Items.AddRange(ВсеЦвета);
            // Можно добавить в список с помощью цикла:
            // foreach (string s in ВсеЦвета)
            // {      listBox1.Items.Add(s); }
            // Сортировать записи в алфавитном порядке
            listBox1.Sorted = true;
        }
        private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
       // Цвет Transparent является "прозрачным", он не поддерживается для
формы:
            if (listBox1.Text == "Transparent") return;
            this.BackColor = Color.FromName(listBox1.Text);
        //this.BackColor = Color.FromName("Blue");
            this.Text = "Цвет: " + listBox1.Text;
        }
    }
}
```

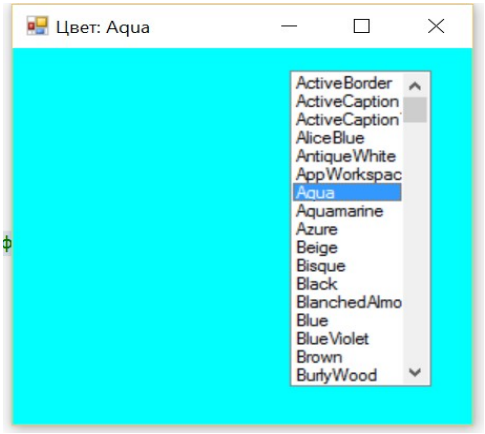


Рис. 4.16. Замалювання форми обраним кольором

Приклад 15.

У цьому прикладі розглядається робота з файлами та діалогами з використанням класів *DirectoryInfo* та *FileInfo*. У програмі використовуються два *ListBox*, які відповідно відображають каталоги та файли. До опису прикладу доцільно розглянути класи *DirectoryInfo* та *FileInfo*.

Клас DirectoryInfo

Цей клас надає методи для створення, видалення, переміщення та інших операцій з каталогами. Деякі з його властивостей та методів:

- Create(): створює каталог
- CreateSubdirectory(path): створює підкаталог по вказаному шляху path
- Delete(): видаляє каталог
- Властивість Exists: визначає, чи існує каталог
- GetDirectories(): отримує список каталогів
- GetFiles(): отримує список файлів
- MoveTo(destDirName): переміщує каталог
- Властивість Parent: отримання батьківського каталогу
- Властивість Root: отримання кореневого каталогу

FileInfo

Для роботи з файлами призначений клас *FileInfo*. З їхньою допомогою ми можемо створювати, видаляти, переміщувати файли, отримувати їх властивості та багато іншого.

Деякі корисні методи та властивості класу *FileInfo*:

- CopyTo(path): копіює файл у нове місце за вказаним шляхом path
- Create(): створює файл
- Delete(): видаляє файл
- MoveTo(destFileName): переміщує файл у нове місце
- Властивість Directory: отримує батьківський каталог як об’єкт DirectoryInfo
- Властивість DirectoryName: отримує повний шлях до батьківського каталогу
- Властивість Exists: вказує, чи існує файл
- Властивість Length: отримує розмір файлу
- Властивість Extension: отримує розширення файлу
- Властивість Name: отримує ім’я файлу
- Властивість FullName: отримує повне ім’я файлу

Клас *File* реалізовує схожу функціональність за допомогою статичних методів:

- Copy(): копіює файл у нове місце
- Create(): створює файл
- Delete(): видаляє файл
- Move: переміщує файл у нове місце
- Exists(file): визначає, чи існує файл

Частина постановки завдання. Введення каталогу в *TextBox*, виведення папок (піддіалогів) у першому *ListBox*. При "кліканні" папки у першому *ListBox* у другому *ListBox* виводяться файли у цій папці. При виділенні файлу у другому *ListBox* у текстових файлах виводяться його характеристики: ім’я, розмір у байтах, дата створення, дата модифікації, дата останнього доступу. Надається можливість щодо обраного каталогу переходу на батьківський каталог.

Текст програми наведено у лістингу 4.15.

Лістинг 4.15. (examp15)

```
namespace examp15
{
    public partial class Form1 : Form
    {
        private string currentFolderPath;

        public Form1()
        {
            InitializeComponent();
        }

        protected void ClearAllFields()
        {
            listBoxFolders.Items.Clear();
            listBoxFiles.Items.Clear();
            txtBoxFolder.Text = "";
            txtBoxFileName.Text = "";
            txtBoxCreationTime.Text = "";
            txtBoxLastAccessTime.Text = "";
            txtBoxLastWriteTime.Text = "";
            txtBoxFileSize.Text = "";
        }

        protected void DisplayFileInfo(string fileFullName)
        {
            FileInfo theFile = new FileInfo(fileFullName);
            if (!theFile.Exists)
            throw new FileNotFoundException("File not found: " + fileFullName);
            txtBoxFileName.Text = theFile.Name;
            txtBoxCreationTime.Text = theFile.CreationTime.ToLongTimeString();
            txtBoxLastAccessTime.Text =
theFile.LastAccessTime.ToLongDateString();
            txtBoxLastWriteTime.Text =
theFile.LastWriteTime.ToLongDateString();
            txtBoxFileSize.Text = theFile.Length + " bytes";
        }

        protected void DisplayFolderList(string folderFullName)
        {
            DirectoryInfo theFolder = new DirectoryInfo(folderFullName);
            if (!theFolder.Exists)
                throw new DirectoryNotFoundException("Folder not found: "
                                                    + folderFullName);
            ClearAllFields();
            txtBoxFolder.Text = theFolder.FullName;
            currentFolderPath = theFolder.FullName;

            // list all subfolders in folder
```

```
                foreach (DirectoryInfo nextFolder in theFolder.GetDirectories())
                    listBoxFolders.Items.Add(nextFolder.Name);

                // list all files in folder
                foreach (FileInfo nextFile in theFolder.GetFiles())
                    listBoxFiles.Items.Add(nextFile.Name);
            }

            protected void OnDisplayButtonClick(object sender, EventArgs e)
            {
                try
                {
                    string folderPath = txtBoxInput.Text;
                    DirectoryInfo theFolder = new DirectoryInfo(folderPath);
                    if (theFolder.Exists)
                    {
                        DisplayFolderList(theFolder.FullName);
                        return;
                    }
                    FileInfo theFile = new FileInfo(folderPath);
                    if (theFile.Exists)
                    {
                        DisplayFolderList(theFile.Directory.FullName);
                        int index = listBoxFiles.Items.IndexOf(theFile.Name);
                        listBoxFiles.SetSelected(index, true);
                        return;
                    }
                    throw new FileNotFoundException("There is no file or folder
with "
                    + "this name: " + txtBoxInput.Text);
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }

            protected void OnListBoxFilesSelected(object sender, EventArgs e)
            {
                try
                {
                    string selectedString = listBoxFiles.SelectedItem.ToString();
                  string fullFileName = Path.Combine(currentFolderPath,
selectedString);
                    DisplayFileInfo(fullFileName);
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }

            protected void OnListBoxFoldersSelected(object sender, EventArgs e)
            {
                try
                {
                    string selectedString = listBoxFolders.SelectedItem.ToString();
                 string fullPathName = Path.Combine(currentFolderPath,
selectedString);
                    DisplayFolderList(fullPathName);
                }
                catch (Exception ex)
```

```
                {
                    MessageBox.Show(ex.Message);
                }
            }
            protected void OnUpButtonClick(object sender, EventArgs e)
            {
                try
                {
                    string folderPath = new
FileInfo(currentFolderPath).DirectoryName;
                    DisplayFolderList(folderPath);
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
        }
    }
```

Розглянемо деякі фрагменти програми.

Щоб отримати список усіх підкаталогів заданого каталогу, ми використовуємо метод *GetDirectories* класу *DirectoryInfo*. Нижче наведено фрагмент коду, що дозволяє записати в перший *ListBox* (*listBoxFolders*) інформацію про всі підкаталоги кореневого каталогу диска *C*:

```
string folderFullName = "C:\\";
DirectoryInfo theFolder = new DirectoryInfo(folderFullName);
// list all subfolders in folder
foreach (DirectoryInfo nextFolder in theFolder.GetDirectories())
listBoxFolders.Items.Add(nextFolder.Name);
```

Заповнення *ListBox* (*listBoxFiles*) файлами у заданому каталозі (theFolder) виконується наступним фрагментом програми:

```
// list all files in folder
foreach (FileInfo nextFile in theFolder.GetFiles())
listBoxFiles.Items.Add(nextFile.Name);
```

При натисканні кнопки *Up* виконується перехід на батьківський каталог. Це виконується властивістю *DirectoryName* класу *FleInfo*, тобто. отримується рядок, що є повним шляхом до батьківського каталогу. Потім *ListBox* заповнюється папками батьківського каталогу.

```
string folderPath = new FileInfo(currentFolderPath).DirectoryName;
DisplayFolderList(folderPath);
```

Фрагмент роботи програми наведено на рис. 4.17.

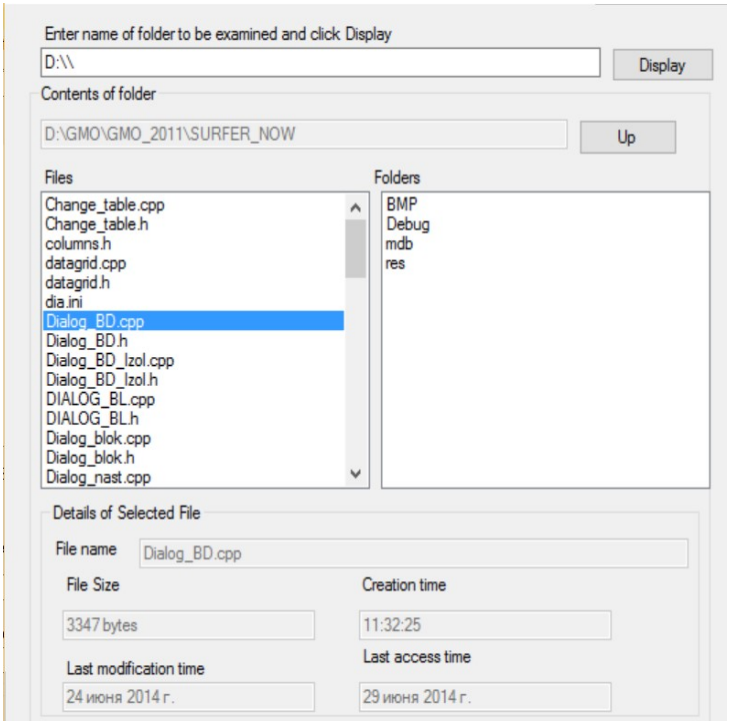


Рис. 4.17. Робота з директоріями та файлами

ListBox зі списком об’єктів

Необхідно відзначити деякі особливості під час роботи *ListBox* зі списком об’єктів.

Оскільки колекція зберігає об’єкти, будь-який коректний тип *.NET* може бути доданий до списку. Для того, щоб ідентифікувати елементи, необхідно встановити дві важливі властивості. Перша з них – *DisplayMember*. Ця установка повідомляє *ListControl*, яка властивість вашого об’єкта має бути відображена у списку. Друга - *ValueMember*, вона вказує властивість вашого об’єкта, яку потрібно повернути як її значення. Якщо до списку додаються рядки, то за замовчуванням вони використовуються для обох цих властивостей. У прикладі як дані списку застосовуються об’єкти *Vendor*. Об’єкт *Vendor* включає лише дві властивості: *Name* і *PhoneNo*. Властивість *DisplayMember* встановлено як *Name*. Це змушує обліковий елемент керування відображати значення властивостей *Name* об’єктів, що містяться в ньому.

Існують два способи доступу до даних у списковому елементі керування, як показано в наступному прикладі коду. Список завантажується об’єктами *Vendor*. Встановлюються властивості *DisplayMember* та *ValueMember*. На початку йде метод *LoadList*. Цей метод завантажує список об’єктів *Vendor* або простих рядків, що містять ім’я постачальника. Кнопка-перемикач використовується для вибору того, що потрібно завантажувати в список.

```
private void LoadList(Control ctrlToLoad) {
    ListBox tmpCtrl = null;
    if (ctrlToLoad is ListBox) tmpCtrl = (ListBox)ctrlToLoad;
    tmpCtrl.Items.Clear();
```

```
 if (radioButton1.Checked) {
// загрузить объекты
tmpCtrl.Items.Add(new Vendor("XYZ Company", "555-555-1234"));
tmpCtrl.Items.Add(new Vendor("ABC Company", "555-555-2345"));
tmpCtrl.Items.Add(new Vendor("Other Company", "555-555-3456"));
tmpCtrl.Items.Add(new Vendor("Another Company", "555-555-4567"));
tmpCtrl.Items.Add(new Vendor("More Company", "555-555-6789"));
tmpCtrl.Items.Add(new Vendor("Last Company", "555-555-7890"));
tmpCtrl.DisplayMember = "Name";
}
else
{
tmpCtrl.Items.Clear();
tmpCtrl.Items.Add("XYZ Company");
tmpCtrl.Items.Add("ABC Company");
tmpCtrl.Items.Add("Other Company");
tmpCtrl.Items.Add("Another Company");
tmpCtrl.Items.Add("More Company");
tmpCtrl.Items.Add("Last Company");
 }
}
```

Після того, як дані було завантажено в список, для їх отримання можна використовувати властивості *SelectedItem* і *SelectedIndex*. Властивість *SelectedItem* повертає поточний вибраний об'єкт. Якщо список дозволяє множинний вибір, немає гарантії того, який елемент буде повернуто. У цьому випадку має використовуватись колекція *SelectObject*. Вона містить список усіх поточних вибраних елементів списку. Якщо потрібно отримати елемент за певним індексом, то властивість *Items* може бути використана для доступу до *ListBox.ObjectCollection*. Оскільки це стандартний клас колекції *.NET*, елементи колекції можна отримати так само, як елементи будь-якого класу колекції. Якщо для наповнення списку використовується *DataBinding* (прив'язка даних), то властивість *SelectedValue* поверне ту властивість вибраного об'єкта, яка була вказана через властивість *ValueMember*. Якщо *ValueMember* встановлено на *Phone*, то *SelectedValue* поверне значення *Phone* вибраного на даний момент елемента списку. Для того, щоб можна було використовувати *ValueMember* і *SelectedValue*, список повинен бути завантажений способом *DataSource*. *ArrayList* або будь-яка інша заснована на *IList* колекція має бути спочатку заповнена об'єктами, потім готовий список може бути наданий властивості *DataSource*. Наступний короткий приклад демонструє описане.

```
listBox1.DataSource = null;
System.Collections.ArrayList lst = new System.Collections.ArrayList();
lst.Add(new Vendor("XYZ Company", "555-555-1234"));
lst.Add(new Vendor("ABC Company", "555-555-2345"));
lst.Add(new Vendor("Other Company", "555-555-3456"));
lst.Add(new Vendor("Another Company", "555-555-4567"));
lst.Add(new Vendor("More Company", "555-555-6789"));
lst.Add(new Vendor("Last Company", "555-555-7890"));
listBox1.Items.Clear();
listBox1.DataSource = lst;
listBox1.DisplayMember = "Name";
listBox1.ValueMember = "Phone";
```

Використання *SelectedValue* без *DataBinding* призведе до виключення *NullException*. У наступному фрагменті коду показано синтаксис доступу до даних списку.

```
// obj установлен в ссылку на выбранный объект Vendor
obj = listBox1.SelectedItem;
// obj установлен в ссылку на объект Vendor по индексу 3 (4-й по счету объект)
obj = listBox.Items[3];
// obj установлен в значение свойства Phone выбранного объекта поставщика
// в этом примере предполагается, что для наполнения списка
// использовалась привязка данных listBox1.ValuesMember = "Phone";
obj = listBox1.SelectValue;
```

Розглянемо два однотипних приклади (приклад 16 та приклад 17) використання *ListBox* зі списками об’єктів.

Приклад 16.

Програму цього прикладу наведено у лістингу 4.16. Фрагмент роботи програми наведено на рис. 4.18.

Лістинг 4.16. (examp16)

```
sing System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp16
{
    public partial class Form1 : Form
    {

        public Form1()
        {
            InitializeComponent();
            listBox1.Items.Add((new Bbb("222", 45)).ToString());
            listBox1.Items.Add(new Bbb("111", 41));
            listBox1.Items.Add(new Bbb("000", 40));
            listBox1.Items.Add(new Bbb("555", 53));
            listBox1.Items.Add(new Bbb("666", 22));
        }

        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show(listBox1.SelectedItem.ToString(),
                listBox1.SelectedIndex.ToString() + "  " +
            ((Bbb)listBox1.SelectedItem).ItemData.ToString());
        }
    }


    //Класс с двумя полями для листбокса...
    class Bbb
```

```
    {
        public Bbb(string n, int I)
        {
            name = n;
            ItemData = I;
        }

        public override string ToString() { return name; }

        public string name;
        public int ItemData;
    }
}
```

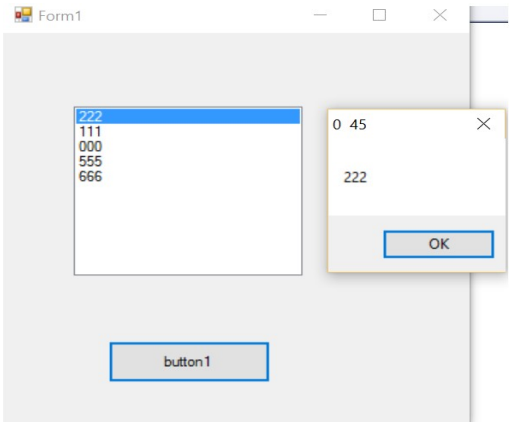


Рис. 4.18. *ListBox* зі списком об’єктів

Приклад 17.

Програму цього прикладу наведено в лістингу 4.17. Фрагмент роботи програми наведено на рис. 4.19.

Лістинг 4.17. (examp17)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp17
{

    public partial class Form1 : Form
    {

        public Form1()
        {
            InitializeComponent();

            List<Phone> phones = new List<Phone>
        {
            new Phone { Id=11, Name="Samsung Galaxy Ace 2", Year=2012},
            new Phone { Id=12, Name="Samsung Galaxy S4", Year=2013},
            new Phone { Id=13, Name="iPhone 6", Year=2014},
            new Phone { Id=14, Name="Microsoft Lumia 435", Year=2015},
            new Phone { Id=15, Name="Xiaomi Mi 5", Year=2015}
```

```
            };

                listBox1.DataSource = phones;
                listBox1.DisplayMember = "Name";
                listBox1.ValueMember = "Id";

                listBox1.SelectedIndexChanged += listBox1_SelectedIndexChanged;
            }

            void listBox1_SelectedIndexChanged(object sender, EventArgs e)
            {
                // получаем id выделенного объекта
                int id = (int)listBox1.SelectedValue;

                // получаем весь выделенный объект
                Phone phone = (Phone)listBox1.SelectedItem;
                MessageBox.Show(id.ToString() + ". " + phone.Name);
            }
        }

        class Phone
        {
            public int Id { get; set; }
            public string Name { get; set; }
            public int Year { get; set; }
        }
}
```

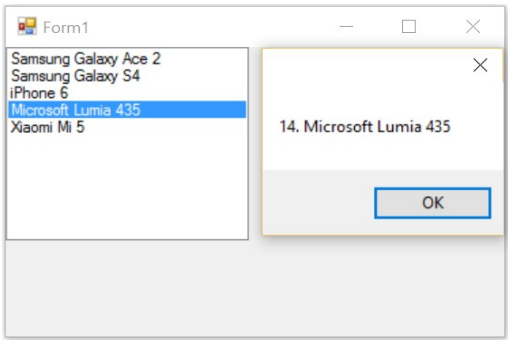


Рис. 4.19. *ListBox* зі списком даних по пристроям

Приклад 18.

Цей приклад більш складніший при використанні об’єкта *ListBox*.

Створимо клас *Deck* (Колода) для зберігання довільної кількості карток, але в класі може бути будь-яка кількість карток (або не бути жодної). Потім у форму, як видно на рис. 4.20, поміщено два *ListBox*, що містять дані по одному об’єкту *Deck* (колод карт). При першому запуску програми в колоді #1 (*ListBox* зліва) може бути до 10 випадкових карток, а в колоді #2 - повний набір - 52 картки. Обидві колоди відсортовані по масті та старшинству. У це положення колоду можна повернути натисканням по кнопці *Reset*. Також форма має кнопки << і >>, які переміщують карти з однієї колоди в іншу. При натисканні кнопки *Shuffle* виконується перетасовування карток випадковим чином. Текст програми наведено у лістингу 4.18.

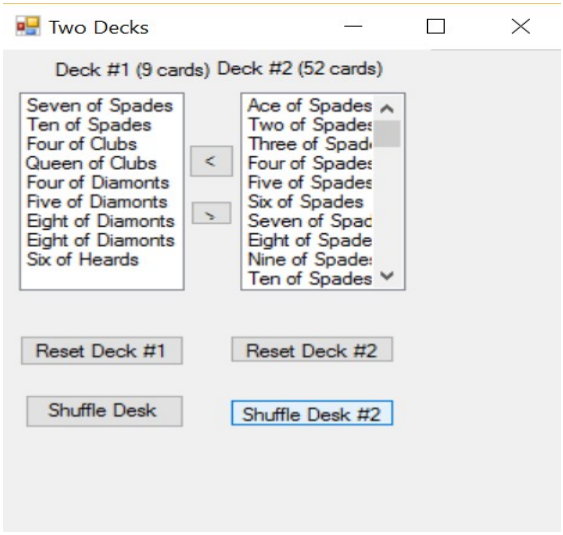


Рис. 4.20. Переміщення карт між колодами

Лістинг 4.18. (examp18)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace ot18
{
    enum Suits // масть
    {
        Spades,
        Clubs,
        Diamonts,
        Heards
    }

    enum Values
    {
        Ace = 1, // карта
        Two = 2,
        Three = 3,
        Four = 4,
        Five = 5,
        Six = 6,
        Seven = 7,
        Eight = 8,
        Nine = 9,
        Ten = 10,
        Jack = 11,
        Queen = 12,
        King = 13
    }
    // ------------------------------------------------------------

    public partial class Form1 : Form
    {
        Deck deck1;
        Deck deck2;
```

```
Random random = new Random();
public Form1()
{
    InitializeComponent();
    ResetDeck(1);
    ResetDeck(2);
    RedrawDeck(1);
    RedrawDeck(2);
    Text = "Two Decks";
}

private void ResetDeck(int deckNumber)
{
    if (deckNumber == 1)
    {
        int numberOfCards = random.Next(1, 11);
        deck1 = new Deck(new Card[] { });
        for (int i = 0; i < numberOfCards; i++)
            deck1.Add(new Card((Suits)random.Next(4),
                (Values)random.Next(1, 14)));
        deck1.Sort();
    }
    else deck2 = new Deck();
}

private void RedrawDeck(int DeckNumber)
{
    if (DeckNumber == 1)
    {
        listBox1.Items.Clear();
        foreach (string cardName in deck1.GetCardNames())
            listBox1.Items.Add(cardName);
        label1.Text = "Deck #1 (" + deck1.Count + " cards)";

    }
    else
    {
        listBox2.Items.Clear();
        foreach (string cardName in deck2.GetCardNames())
            listBox2.Items.Add(cardName);
        label2.Text = "Deck #2 (" + deck2.Count + " cards)";
    }
}

private void reset1_Click(object sender, EventArgs e)
{
    ResetDeck(1);
    RedrawDeck(1);
}

private void reset2_Click(object sender, EventArgs e)
{
    ResetDeck(2);
    RedrawDeck(2);
}

private void shuffle1_Click(object sender, EventArgs e)
{
    deck1.Shuffle();
    RedrawDeck(1);
}
```

```
        private void shuffle2_Click(object sender, EventArgs e)
        {
            deck2.Shuffle();
            RedrawDeck(2);
        }

        private void moveToDesk1_Click(object sender, EventArgs e)
        {
            if (listBox2.SelectedIndex >= 0)
                if (deck2.Count > 0)
                    deck1.Add(deck2.Deal(listBox2.SelectedIndex));
            RedrawDeck(1);
            RedrawDeck(2);
        }

        private void moveToDeck2_Click(object sender, EventArgs e)
        {
            if (listBox1.SelectedIndex >= 0)
                if (deck1.Count > 0)
                    deck2.Add(deck1.Deal(listBox1.SelectedIndex));
            RedrawDeck(1);
            RedrawDeck(2);
        }
    }

// ------------------------------------------------------------------------
    class Card
    {
        public Suits Suit { get; set; }
        public Values Value { get; set; }
        public Card(Suits suit, Values value)
        {
            this.Value = value;
            this.Suit = suit;
        }

        public override string ToString()
        {
            return Name;
        }

        public string Name
        {
            get { return Value.ToString() + " of " + Suit.ToString(); }
            set { ;}
        }
    }

//--------------

    class Deck //  колода
    {

        private List<Card> cards;
        private Random random = new Random();
        public Deck()
        {
            cards = new List<Card>();
            for (int suit = 0; suit <= 3; suit++)
                for (int value = 1; value <= 13; value++)
                    cards.Add(new Card((Suits)suit, (Values)value));
        }
```

```
        public Deck(IEnumerable<Card> initialCards)
        {
            cards = new List<Card>(initialCards);
        }
        public int Count { get { return cards.Count; } }
        public void Add(Card cardToAdd)
        { cards.Add(cardToAdd); }

        public Card Deal(int index)
        {
            Card CardToDeal = cards[index];
            cards.RemoveAt(index);
            return CardToDeal;
        }
        public void Shuffle()
        {
            List<Card> NewCards = new List<Card>();
            while (cards.Count > 0)
            {
                int CardToMove = random.Next(cards.Count);
                NewCards.Add(cards[CardToMove]);
                cards.RemoveAt(CardToMove);
            }
            cards = NewCards;
        }
        public IEnumerable<string> GetCardNames()
        {
            string[] CardNames = new string[cards.Count];
            for (int i = 0; i < cards.Count; i++)
                CardNames[i] = cards[i].Name;
            return CardNames;
        }
        public void Sort()
        {
            cards.Sort(new CardComparer_bySuit());
        }
    }
    class CardComparer_bySuit : IComparer<Card>
    {
        public int Compare(Card x, Card y)
        {
            if (x.Suit > y.Suit) return 1;
            if (x.Suit < y.Suit) return -1;
            if (x.Value > y.Value) return 1;
            if (x.Value < y.Value) return -1;
            return 0;
        }
    }
}
```

CheckedListBox

Елемент *CheckedListBox* представляє симбіоз компонентів *ListBox* та *CheckBox*. Для кожного елемента такого списку визначено спеціальне поле *CheckBox*, яке можна обрати.

Всі елементи в *CheckedListBox* задаються як *Items*. Також, як і для елементів *ListBox* та *ComboBox*, ми можемо задати набір елементів. За замовчуванням для кожного нового елемента, що додається, прапорець не позначений, як це показано на рис. 4.21.

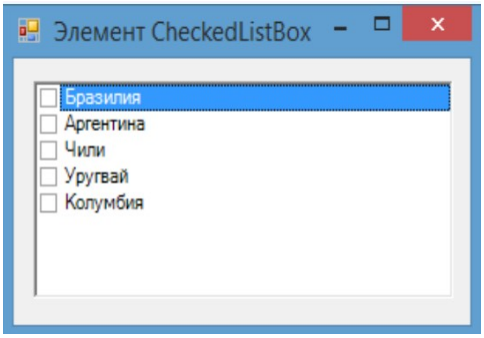


Рис. 4.21. Положення CheckedListBox за замовчуванням.

Щоб поставити позначку в *CheckBox* поряд з елементом у списку, нам треба спочатку виділити елемент і додатковим натисканням вже встановити прапорець. Однак це не завжди зручно, і за допомогою властивості *CheckOnClick* та встановлення для неї значення *true* ми можемо визначити одразу обрання елемента та встановлення для нього прапорця в один клік.

Інша властивість *MultiColumn* при значенні *true* дозволяє зробити список багатостовпчатим, якщо елементи не поміщаються по довжині. Це можна побачити на рис. 4.22.

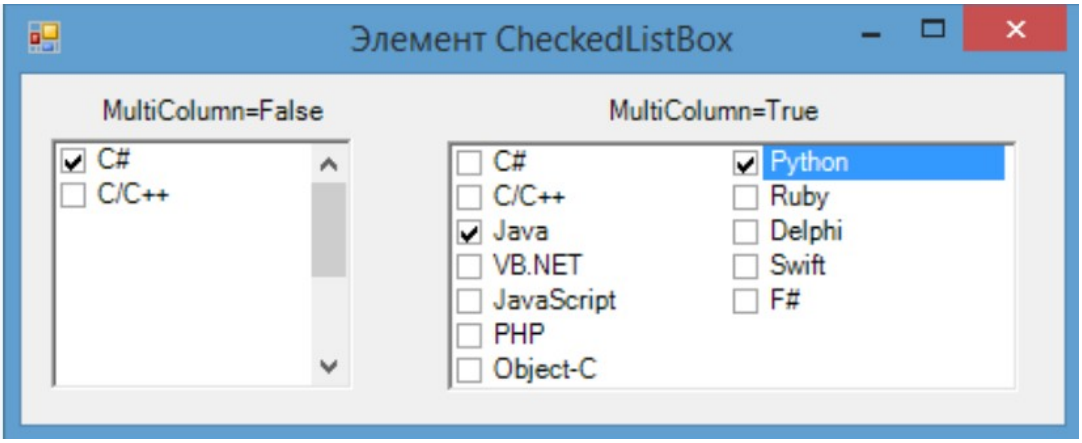


Рис. 4.22. Одно та багатостовпчатий список

Виділений елемент ми також можемо отримати за допомогою властивості *SelectedItem*, а його індекс – за допомогою властивості *SelectedIndex*. Але це правильно тільки, якщо для властивості *SelectionMode* встановлено значення *One*, що передбачає виділення лише одного елемента.

При установці властивості *SelectionMode* значень *MultiSimple* і *MultiExtended* можна вибрати відразу кілька елементів, і тоді всі вибрані елементи будуть доступні у властивості *SelectedItems*, а їх індекси - у властивості *SelectedIndeces*.

І оскільки ми можемо поставити позначку не для всіх вибраних елементів, то щоб окремо отримати зазначені елементи, *CheckedListBox* має властивості *CheckedItems* і *CheckedIndices*.

Для додавання та видалення елементів у *CheckedListBox* визначені ті самі методи, що і в *ListBox*:

- *Add(item)*: додає один елемент
- *AddRange(array)*: додає до списку масив елементів
- *Insert(index, item)*: додає елемент за певним індексом
- *Remove(item)*: видаляє елемент
- *RemoveAt(index)*: видаляє елемент за певним індексом
- *Clear()*: повністю очищає список

До особливостей елемента можна віднести методи *SetItemChecked* та *SetItemCheckState*. Метод *SetItemChecked* дозволяє встановити або прибрати позначку на одному з елементів. А метод *SetItemCheckState* дозволяє встановити прапорець в один із трьох станів: *Checked* (позначено), *Unchecked* (непозначено) та *Indeterminate* (проміжний стан):

```
checkedListBox1.SetItemChecked(0, true);
checkedListBox1.SetItemCheckState(1, CheckState.Indeterminate)
```

На рис. 4.23. наведено приклад установки прапорця в різні стани.

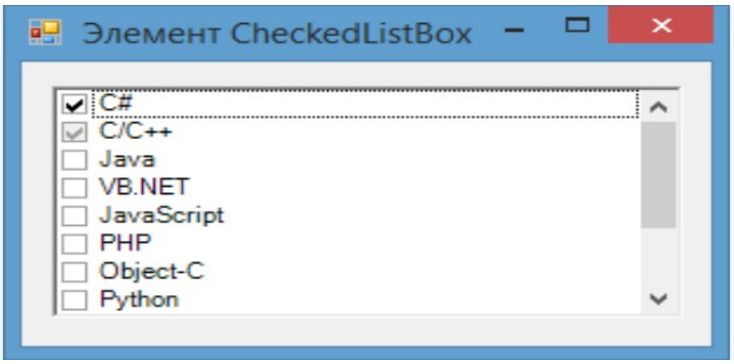


Рис. 4.23. Установка або прибрання відміток у *CheckedListBox*

ComboBox

ComboBox є комбінацією текстового поля що може редагуватися та вікна списку.

За допомогою низки властивостей можна налаштувати стиль оформлення компонента. Так, властивість *DropDownWidth* задає ширину спадаючого списку. За допомогою властивості *DropDownHeight* можна встановити висоту спадаючого списку. Для пошуку використовується метод *FindString* що знаходить перший рядок у списку, який починається з переданого в аргументі фрагмента. Метод *FindStringExact* шукає перший рядок, який буквально відповідає рядку, переданому в аргументі. Обидва способи повертають індекс знайденого значення, або -1, якщо значення знайдено. Вони також можуть приймати додатковий цілий аргумент - стартову позицію пошуку.

Властивість *DropDownStyle* задає стиль *ComboBox*. Вона може приймати три можливі значення:

- *Dropdown*: використовується за замовчуванням. Ми можемо відкрити список спадаючих варіантів, при введенні значення в текстове поле або натиснувши на кнопку зі стрілкою в правій частині

елемента, і нам відобразиться власне список, в якому можна вибрати можливий варіант

- *DropdownList*: щоб відкрити спадаючий список, треба натиснути на кнопку зі стрілкою в правій частині елемента
- *Simple*: *ComboBox* представляє просте текстове поле, в якому ми можемо використовувати клавіші клавіатури вгору/вниз для переходу між елементами.

Залежно від значення *DropDownStyle*, на рис. 4.24 представлено вигляд *ComboBox*.

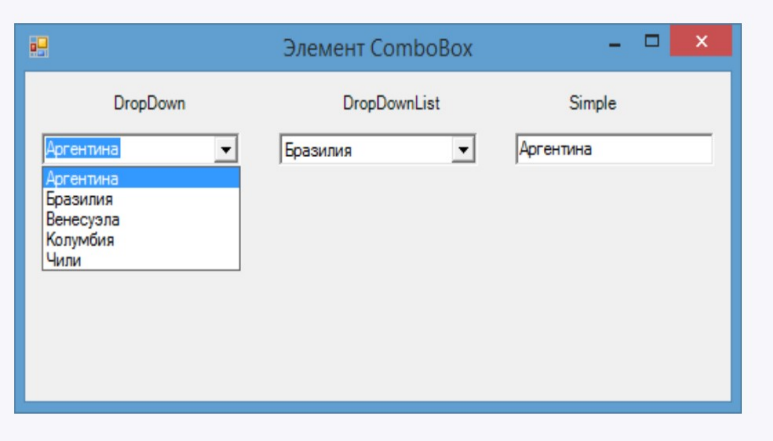


Рис. 4.24. Стилі *ComboBox*

Найбільш важливою подією для *ComboBox* також є подія *SelectedIndexChanged*, що дозволяє відстежити вибір елемента у списку:

```
public partial class Form1 : Form
{
     public Form1()
     {
          InitializeComponent();
         comboBox1.SelectedIndexChanged += comboBox1_SelectedIndexChanged;
     }

     void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
     {
          string selectedState = comboBox1.SelectedItem.ToString();
          MessageBox.Show(selectedState);
     }
}
```

Елемент *ComboBox* нагадує елемент *ListBox*. Він зберігає колекцію об’єктів *ComboBoxItem*, які створюються явним чи неявним чином. Як і *ListBoxItem*, *ComboBoxItem* є елементом керування вмістом, який може зберігати будь-який вкладений елемент. Для заповнення *ComboBox* використовуються способи практично такі ж як і для *ListBox*:

```
1. cmbBox.Items.Add("бананы");
   cmbBox.Items.Add(" киви");

2. string [] fruit = {"апельсины", "яблоки", "виноград",
"манго"};
   cmbBox.Items.AddRange(fruit);

3. List<string> vegetables;
  vegetables = new List<string> {"картошка", "капуста",
"свекла"};
    cmbBox.DataSource = vegetables;
```

Основною відмінністю класів *ComboBox* та *ListBox* є спосіб їх відображення у вікні. Елемент *ComboBox* використовує список, що розкривається, а це означає, що за один раз можна вибрати тільки один елемент.

Якщо потрібно зробити так, щоб користувач міг обрати елемент у *ComboBox*, ввівши текст у текстовому полі, необхідно надати властивості *IsEditable* значення *true*. Крім того, потрібно зберігати лише звичайні текстові об'єкти *ComboBoxItem* або об'єкти з осмисленим поданням *ToString()*. Наприклад, якщо заповнити список, що розкривається, об'єктами *Image*, то текст, який з'явиться у верхній частині, буде повністю визначений ім'ям класу *Image*, а це навряд чи те, що треба. Одним з обмежень елемента *ComboBox* є спосіб підгонки при автоматичному виборі розміру. *ComboBox* вибирає таку ширину, щоб умістити свій вміст, тобто. змінює розмір під час переходу від одного елемента до іншого. На жаль, немає легкого способу вказати *ComboBox* прийняти розмір найбільшого елемента. Натомість доводиться вказувати жорстко закодоване значення властивості *Width*, що дуже незручно.

4.5. Автоматичне масштабування елементів керування

Для перемасштабування елементів керування при зміні властивості *Font* можна використовувати два способи. Перший був використаний для старих версій 1.0, 1.1 .NET Framework. У сучасних версіях підтримується перший спосіб і запропонований новий другий спосіб.

Розглянемо перший метод.

Приклад 19

У лістингу 4.19. та на рис. 4.25. наведено реалізацію перемасштабування кнопок першим способом. Тут створюються п'ять кнопок, які дають змогу вибрати різні розміри шрифту. Конструктор програми використовує розміри та місцезнаходження на основі традиційної системи координат діалогового вікна. Для зберігання цілого значення розміру в пунктах, пов'язаного з кожною кнопкою, в програмі використовується властивість *Tag* елемента керування *Button*.

При натисканні по кнопці масштабування виконується на основі поточного та нового шрифтів за допомогою обробника події *Click*. Тут використовується статичний метод форми *GetAutoScaleSize()*, що визначає ширину та висоту об'єкта *Font*:

```
SizeF GetAutoScaleSize(Font font)
```

Ширина, що повертається, обчислюється як середня на основі символів латинського алфавіту в нижньому регістрі. Висота ж дорівнює властивості *Height* об'єкта *Font*.

Зрештою, перемасштабування виконується з використанням методу *Scale*. Синтаксис методу:

```
void Scale(float xScale, yScale)
```

Лістинг 4.19. (examp19)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp19
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            Text = "Auto-Scale Demo";
            Font = new Font("Arial", 12);
            AutoScaleMode = AutoScaleMode.Font;
            FormBorderStyle = FormBorderStyle.FixedSingle;
            int[] Pt = { 8, 12, 18, 24, 32 };
            for (int i = 0; i < Pt.Length; i++)
            {
                Button btn = new Button();
                btn.Parent = this;
                btn.Text = "Use " + Pt[i] + "-point font";
                btn.Tag = Pt[i];
                btn.Location = new Point(4, 18 + 40 * i);
                btn.Size = new Size(180, 30);
                btn.Click += new EventHandler(ButtonOnClick);
            }
        }

        void ButtonOnClick(object obj, EventArgs ea)
        {
            Button btn = (Button)obj;
            // Получение старых размеров
            SizeF sizefOld = GetAutoScaleSize(Font);
            Font = new Font(Font.FontFamily, (int)btn.Tag);
            // Получение новых размеров
            SizeF sizefNew = GetAutoScaleSize(Font);
```

```
            //Изменение масштабирования
            Scale(sizefNew.Width / sizefOld.Width,
            sizefNew.Height / sizefOld.Height);
        }
        protected override void OnPaint(PaintEventArgs e)
        {
            base.OnPaint(e);
            e.Graphics.DrawString(Text, Font, new SolidBrush(ForeColor), 0, 0);
        }
    }
}
```

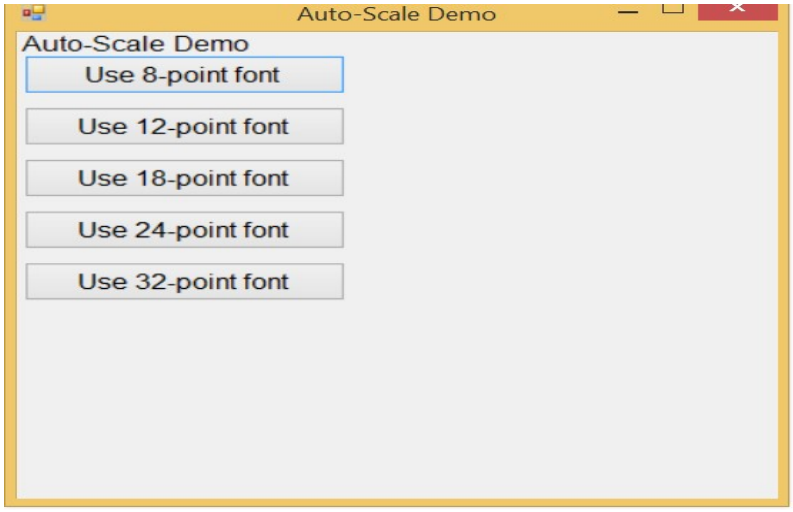


Рис. 4.25. До перемасштабування елементів керування

Приклад 20.

У цьому прикладі реалізується другий спосіб перемасштабування кнопок, який значно простіше за перший спосіб. Достатньо встановити режим масштабування властивістю форми *AutoScaleMode = <Режим масштабування>*.

Режим масштабування встановлюється переліками (типу *enum*), що наведені в табл. 4.3. За замовчуванням - режим масштабування *None*. У більшості бізнес-додатків використовується режим масштабування *Font*. Саме цей режим використовується для реалізації другого способу, наведеного в лістингу 4.20.

Таблиця 4.3

Режими масштабування

	Режим масштабування	**Опис**
1	*Dpi*	Режим масштабування корисний для графічних програм і сумісний з масштабуванням за замовчуванням, яке використовується .NET Compact Framework.
2	*Font*	Масштаб елементів керування щодо розмірів шрифту, який використовується класами, що зазвичай є системним шрифтом.
3	*Inherit*	Масштаб елементів керування відповідно до батьківського режиму масштабування класів. Якщо батьківський об’єкт відсутній, автоматичне масштабування вимкнено.
4	*None*	Автоматичне масштабування вимкнено.

Лістинг 4.20. (examp20)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp20
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            Text = "Auto-Scale Demo";
            Font = new Font("Arial", 12);

            AutoScaleMode = AutoScaleMode.Font;
            FormBorderStyle = FormBorderStyle.FixedSingle;
            int[] Pt = { 8, 12, 18, 24, 32 };
            for (int i = 0; i < Pt.Length; i++)
            {
                Button btn = new Button();
                btn.Parent = this;
                btn.Text = "Use " + Pt[i] + "-point font";
                btn.Tag = Pt[i];
                btn.Location = new Point(4, 18 + 40 * i);
                btn.Size = new Size(180, 30);
                btn.Click += new EventHandler(ButtonOnClick);
            }
        }

        void ButtonOnClick(object obj, EventArgs ea)
        {
            Button btn = (Button)obj;
            Font = new Font(Font.FontFamily, (int)btn.Tag);
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            base.OnPaint(e);
            e.Graphics.DrawString(Text, Font, new SolidBrush(ForeColor), 0, 0);
        }
    }
```

4.6. Смуга прокручування (VScrollBar, HScrollBar, TrackBar)

VScrollBar, HScrollBar

Слід розуміти відмінності між елементом керування "смуга прокручування" та смугами прокручування, які створюються автоматично і є приналежністю будь-якого класу-спадкоємця *ScrollableControl* (наприклад, *Form* і *Panel*). Для автоматичної прокрутки досить встановити властивість *AutoScroll* значення *true*, тобто. смуги прокручування автоматично з’являються праворуч та знизу клієнтської області або панелі. Це має місце в

тому випадку, коли елементи керування знаходяться повністю або частково поза межами клієнтської форми або панелі. Слід зазначити, що з *AutoScroll = true* немає сенсу присвоювати властивостям *HScroll* чи *VScroll* значення *true*. З автоматичними смугами прокручування події не пов'язані.

У розділі ми розглядаємо не автоматичну прокрутку, а елементи керування *VScrollBar* і *HScrollBar*, використовувані окремо, тобто самі по собі. Вони є похідними від класу *ScrollBar*. Вікно елемента керування *HScrollBar* розташовується горизонтально, а елемента керування *VScrollBar* вертикально. Розглянуті елементи керування (користувачі часто називають їх "повзунками") в основному використовуються для вибору значень, які плавно змінюються - наприклад, рівня гучності, яскравості, стиснення, пріоритету і т.п.

Після створення елемента керування необхідно налаштувати його властивості, а також підключити обробник події *Scroll* та (або) *ValueChanged*:

Основні властивості елементів керування *HScrollBar* та *VScrollBar*:

Value - зберігає поточне значення, безпосередньо пов'язане з положенням движка у вікні смуги прокручування. При зміні значення властивості положення движка у вікні смуги прокручування змінюватиметься відповідним чином.

Minimum і *Maximum* визначають, відповідно, мінімальне та максимальне значення, пов'язане зі смугою прокручування. За замовчуванням мінімальне значення 1, а максимальне - 100.

Дискретність зміни значення під час використання клавіш переміщення курсору визначається властивістю *SmallChange*. За замовчуванням вона дорівнює 1. Дискретність під час використання клавіш *PgUp* і *PgDn* задається за допомогою властивості *LargeChange* і дорівнює за замовчуванням 10.

Приклад 21.

На лістингу 4.21. та рис. 4.26. наведено програму *ColorScroll* та її реалізацію. Ця рограма використовує три смуги прокручування: *Red* (Червоний), *Green* (Зелений) та *Blue* (Блакитний), які вибирають поєднання кольорів. У програмі використовуються чотири елементи керування *Label* – об'єкти *label1*, *label2*, *label3*, *label4*. Результуючий колір подається в *label1*. В інших Label виводяться написи зі значеннями частки складових *RGB*.

Лістинг 4.21. (examp21)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp21
{
    //Пример работы с ScrollBar
    public partial class Form1 : Form
```

```
    {
        public Form1()
        {
            InitializeComponent();
            label2.Text = "0";
            label3.Text = "0";
            label4.Text = "0";
            label1.Text = "";

            vScrollBar1.Minimum = 0;
            vScrollBar1.Maximum = 255;
            vScrollBar1.SmallChange = 1;
            vScrollBar1.LargeChange = 10;
            vScrollBar1.Value = 0;

            vScrollBar2.Minimum = 0;
            vScrollBar2.Maximum = 255;
            vScrollBar2.SmallChange = 1;
            vScrollBar2.LargeChange = 10;
            vScrollBar2.Value = 0;

            vScrollBar3.Minimum = 0;
            vScrollBar3.Maximum = 255;
            vScrollBar3.SmallChange = 1;
            vScrollBar3.LargeChange = 10;
            vScrollBar3.Value = 0;
            label1.BackColor = Color.FromArgb(
            vScrollBar1.Value,vScrollBar2.Value,vScrollBar3.Value);
        }

        private void vScrollBar1_Scroll(object sender, ScrollEventArgs e)
        {
            label1.BackColor = Color.FromArgb(
            vScrollBar1.Value, vScrollBar2.Value, vScrollBar3.Value);
            label2.Text = vScrollBar1.Value.ToString();
            label3.Text = vScrollBar2.Value.ToString();
            label4.Text = vScrollBar3.Value.ToString();
            VScrollBar vv = (VScrollBar)sender;
            label1.Text = vv.Name;
        }
    }
}
```

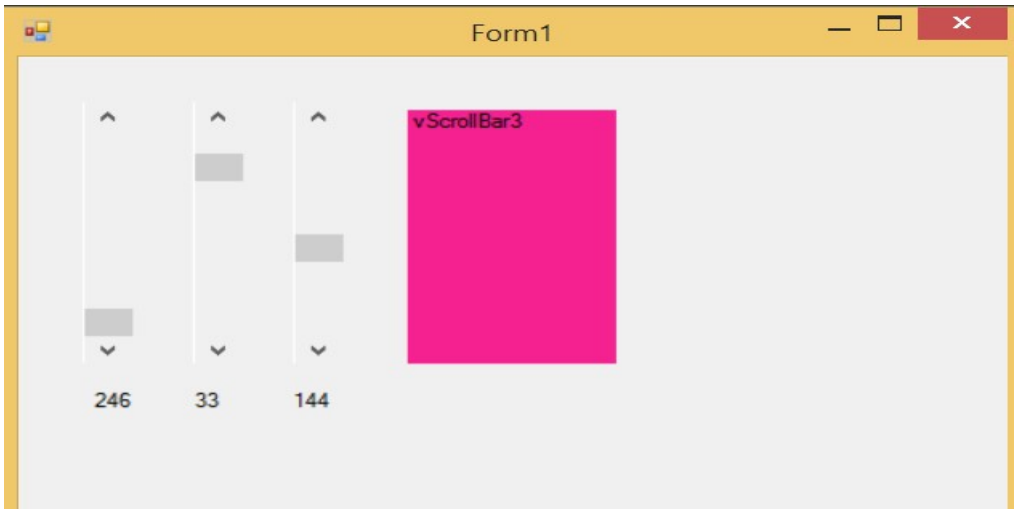


Рис. 4.26. Використання VScrollBar

Багато спільного із елементом *ScrollBar* має елемент *SpinButton*. Елемент керування *SpinButton* - це та сама смуга прокручування, позбавлена самої смуги та повзунка. *SpinButton* використовується в тих ситуаціях, коли діапазон значень, що вибираються, зовсім невеликий (наприклад, треба вибрати кількість копій для друку звіту). Усі властивості, які є у *SpinButton*, збігаються з властивостями *ScrollBar*. Вигляд лічильника наведено на рис. 4.27.

Рис. 4.27. Об’єкт SpinButton (накопичувач)

TrackBar.

Елемент керування *TrackBar* має таке призначення як і *SCrollBar*, тому багато властивостей у них однакові. У цьому є додаткові характеристики. Крім двигуна, у вікні елемента керування *TrackBar* є розділювачі. Вони відображаються у вигляді коротких штрихів, що розташовані на рівній відстані один від одного. Щоб задати кількість штрихів, які відображаються на шкалі елемента керування *TrackBar*, потрібно відредагувати властивість *TickFrequency*. Змінюючи властивість Orientation, можна встановити горизонтальне або вертикальне розташування вікна елемента керування *TrackBar*. У першому випадку властивість має містити значення *Orientation.Horizontal*, а в другому - *Orientation.Vertical*.

Властивість *TickStyle* задає стиль шкали та повзунка. Ось можливі значення: *None*, *TopLeft*, *BottomRight*, *Both*.

У першому випадку при використанні значення *None* штрихи не відображаються на шкалі. Інші константи дозволяють задати розташування штрихів зверху або знизу (праворуч або ліворуч) від движка, а також по обидва боки движка (значення *Both*).

На лістингу 4.22. наведено програму, яка за постановкою така сама, як і попередня. Тут замість *VScrollBar* використовується *TrackBar*. На рис. 4.28. показано реалізацію програми.

Приклад 22.

Лістинг 4.22. (examp22)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
namespace exam22
{
    //Пример работы с trackBar
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            label2.Text = "0";
            label3.Text = "0";
            label4.Text = "0";
            label1.Text = "";

            trackBar1.Minimum = 0;
            trackBar1.Maximum = 255;
            trackBar1.SmallChange = 1;
            trackBar1.LargeChange = 10;
            trackBar1.Value = 0;
            trackBar1.TickStyle = TickStyle.Both;
            trackBar1.Orientation = Orientation.Vertical;
            trackBar1.TickFrequency = 25;

            trackBar2.Minimum = 0;
            trackBar2.Maximum = 255;
            trackBar2.SmallChange = 1;
            trackBar2.LargeChange = 10;
            trackBar2.Value = 0;
            trackBar1.TickStyle = TickStyle.Both;
            trackBar1.Orientation = Orientation.Vertical;
            trackBar3.TickFrequency = 25;

            trackBar3.Minimum = 0;
            trackBar3.Maximum = 255;
            trackBar3.SmallChange = 1;
            trackBar3.LargeChange = 10;
            trackBar3.Value = 0;
            trackBar3.TickStyle = TickStyle.BottomRight;
            trackBar1.Orientation = Orientation.Vertical;
            trackBar3.TickFrequency = 25;
            label1.BackColor = Color.FromArgb(
            trackBar1.Value,trackBar2.Value,trackBar3.Value);
        }

        private void trackBar1_Scroll(object sender, EventArgs e)
        {
            label1.BackColor = Color.FromArgb(
            trackBar1.Value, trackBar2.Value, trackBar3.Value);
            label2.Text = trackBar1.Value.ToString();
            label3.Text = trackBar2.Value.ToString();
            label4.Text = trackBar3.Value.ToString();
            TrackBar vv = (TrackBar)sender;
            label1.Text = vv.Name;
        }
    }
}
```

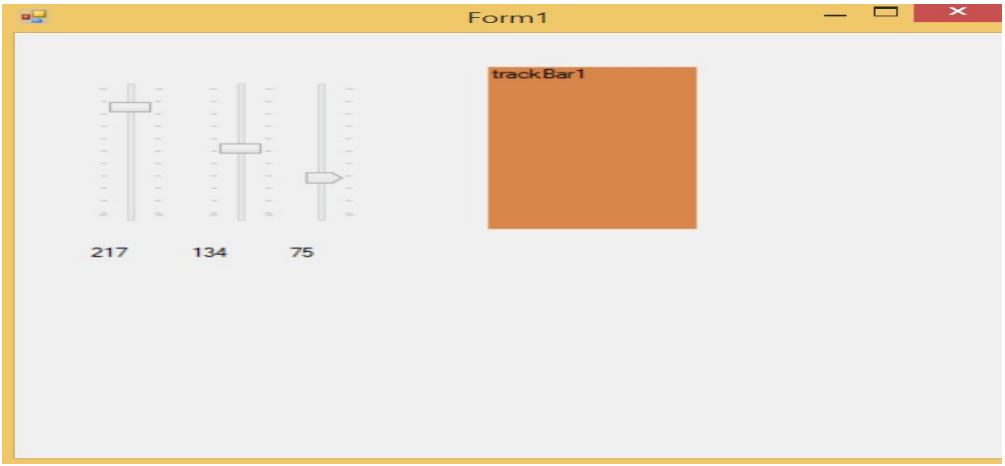


Рис. 4.28. Використання TrackBar

4.7. Елемент керування Timer. Програмування годинників на основі кривих Безьє

Timer є компонентом запуску дій, повторюваних через певний проміжок часу. Хоча він не є візуальним елементом, але його також можна перетягнути з панелі інструментів на форму.

Найбільш важливі властивості та методи таймера:

- Властивість *Enabled*: при значенні *true* вказує, що таймер запускатиметься разом із запуском форми
- Властивість *Interval*: вказує інтервал у мілісекундах, через який спрацьовуватиме обробник події *Tick*, яка має таймер.
- Метод *Start()*: запускає таймер
- Метод *Stop()*: зупиняє таймер

Приклад 23

Наприклад визначимо просту форму, на яку додамо кнопку і таймер. У файлі коду форми визначимо код, поданий лістингом 4.23. Фрагмент виконання програми наведено на рис. 4.29.

Лістинг 4.23. (examp23)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp23
{
    public partial class Form1 : Form {
        int koef = 1;
        public Form1()
        {
            InitializeComponent();
            this.Width = 400;
```

```
            button1.Width = 40;
            button1.Left = 40;
            button1.Text = "";
            button1.BackColor = Color.Aqua;
            timer1.Interval = 500; // 500 миллисекунд
            timer1.Enabled = true;
            button1.Click += button1_Click;
            timer1.Tick += timer1_Tick;
        }
        // обработчик события Tick таймера
        void timer1_Tick(object sender, EventArgs e)
        {
            if (button1.Left == (this.Width - button1.Width - 10))
            {
                koef = -1;
            }
            else if (button1.Left == 0)
            {
                koef = 1;
            }
            button1.Left += 10 * koef;
        }
        // обработчик нажатия на кнопку
        void button1_Click(object sender, EventArgs e)
        {
            if (timer1.Enabled == true) {
                timer1.Stop();
            }
            else
            {
                timer1.Start();
            }
        }
    }
}
```

У конструкторі форми встановлюються початкові значення для таймера, кнопки та форми. Через кожен інтервал таймера спрацьовуватиме обробник *timer1_Tick*, в якому змінюється положення кнопки по горизонталі за допомогою властивості *button1.Left*. А за допомогою додаткової змінної *koef* можна керувати напрямком руху.

Крім того, за допомогою обробника натискання кнопки *button1_Click* можна або зупинити таймер (та разом з ним і рух кнопки), або знову його запустити.

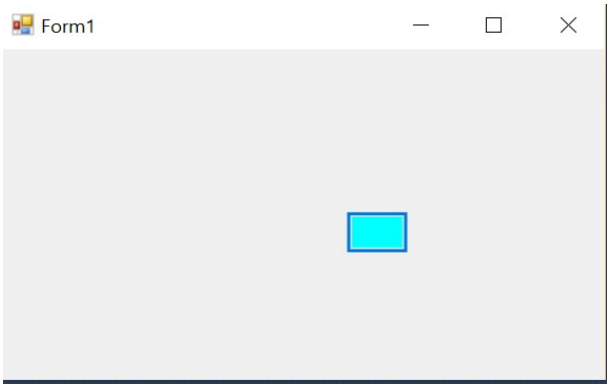


Рис. 4.29. Переміщення кнопки

Програмування годинників на основі кривих Безьє

Криві Безье були розроблені в 60-х роках XX століття незалежно один від одного П'єром Безье. Незважаючи на те, що відкриття де Кастельжо було зроблено раніше Без'є (1959), його дослідження не публікувалися і ховалися компанією як виробнича таємниця до кінця 1960-х. Згодом це відкриття стало одним із найважливіших інструментів систем автоматизованого проектування та програм комп'ютерної графіки.

Залежно від кількості контрольних точок N будується згладжена крива ступеню:

$$L = N\text{-}1.$$

При $N = 2$ крива є відрізком прямої лінії, опорні точки $\mathbf{P_0}$ і $\mathbf{P_1}$ визначають його початок і кінець. Крива задається параметричним рівнянням:

$$P(t) = (1 - t)P_0 + tP_1, \text{ где}$$

P – це параметричний показник, тобто X або Y; t – змінюється від 0 до 1

При $N = 3$ ($L = 2$) буде квадратична крива, опорні точки $\mathbf{P}_0$, $\mathbf{P}_1$ та $\mathbf{P}_2$ визначають її початок та кінець. Криву наведено на рис. 4.30. Її параметричне рівняння:

$$P(t) = (1 - t)^2P_0 + (1 - t)tP_1 + t^2P_2, \qquad 0 \le t \le 1$$

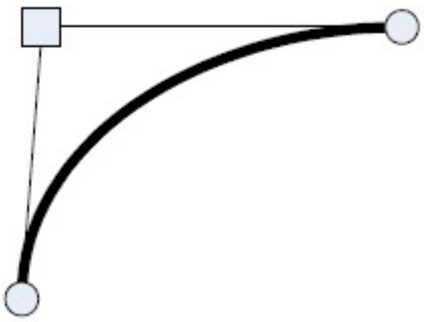


Рис 4.30. Крива Безьє за трьома опорними точками.

При N = 4 кубічна крива Безьє (L = 3, см. рис. 4.31.) описується наступним рівнянням:

$$P(t) = (1 - t)^3P_0 + 3t(1 - t)^2P_2 + 3t^2(1 - t)P_2 + t^3P_3,$$

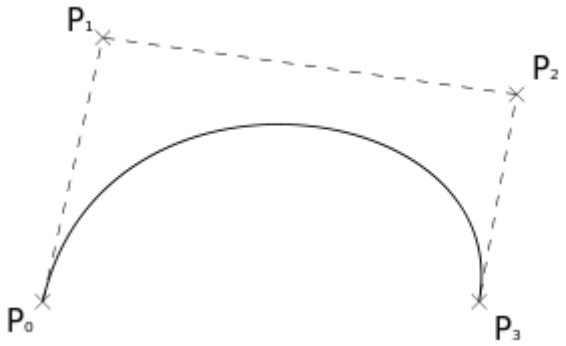


Рис. 4.31. Кубічна крива Безьє (за 4 опорними точкам)

Як бачимо, зі збільшенням кількості опорних точок, зростає ступінь рівняння. Це є великим недоліком, оскільки можна вийти за межі значимості чисел. Безумовно це враховано при розробці стандартних функцій малювання кривих Безье. У *C#* є два методи *DrawBezier* і *DrawBeziers*:

```
public void DrawBezier(Pen, Point, Point, Point, Point);
public void DrawBezier(Pen, PointF, PointF, PointF, PointF);
public void DrawBezier(Pen, float, float, float, float, float, float,
float, float);

public void DrawBeziers(Pen, Point[]);
public void DrawBeziers(Pen, PointF[]);
```

У всіх цих методах перший параметр визначає перо, яке буде використано для малювання. Інші параметри задають координати опорних точок. У методі *DrawBezier* використовується строго 4 точки. Що стосується методу *DrawBeziers*, то він дозволяє задавати координати точок у вигляді масивів без обмежень кількості точок. Тут передаються точки для побудови сегментів М кривої. Тобто сумарна крива будується із окремих сегментів. Кількість точок у масиві має бути кратним 3 плюс 1, оскільки для першої кривої використовується 4 крапки, і кожен із решти сплайнів вимагає по 3 крапки. Перша крива Безьє малюється від першої точки до четвертої точки в масиві крапок. Кожна наступна крива потребує рівно три точки. Кінцева точка попередньої кривої використовується як перша точка для кожної наступної кривої. При кількості точок відмінного від $3 * n + 1$, де $n = 1,2 \ldots m$ крива не буде будуватися.

Приклад 24.

Крива Безьє використовується у програмі, наведеній на лістингу 4.24. У цій програмі реалізовано годинник. Відображення годинника виконується у вигляді окремого розробленого дочірнього елемента керування *ClockControl* вікна *Frame1*. При створенні власних елементів керування рекомендується використовувати батьківський клас *UserControl*, який успадковується від класу *Control*, який у свою чергу успадковується від класів *ScrollableControl* і *ContainerControl*.

У загальному вигляді *UserControl* це по своїй суті "контроль", на якому можна групувати стандартні елементи, мати свій *Paint*, властивості, методи, відгуки.

Пристосовуючи до нашого випадку, використання *UserControl* полегшує введення годинника в іншу програму або написання програми, що відображає безліч циферблатів. Можна використовувати колір форми та розробленого дочірнього елемента (об’єкта).

Лістинг 4.24. (examp24)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp24
{
    public partial class Form1 : Form
    {
        ClockControl clock;
        // Пример работы с часами (стрелка часов рисуется с использованием
Безье)

        public Form1()
        {
           InitializeComponent();
           Text = "Analog  Clock";
           BackColor = SystemColors.Window;
           ForeColor = SystemColors.WindowText;
           timer1.Interval = 1000;
           timer1.Start();
.....ResizeRedraw = true;
.....ClientSize = new System.Drawing.Size(500, 400);
           clock = new ClockControl();
           clock.ClientSize = new System.Drawing.Size(400, 300);
           clock.Parent = this;
           clock.Time = DateTime.Now;
           clock.Dock = DockStyle.Fill;
           clock.BackColor = Color.Black;
           clock.ForeColor = Color.White;
        }
        private void timer1_Tick(object sender, EventArgs e)
        {
            clock.Time = DateTime.Now;
        }
    }

// ---------------------------------------------------------------------
    class ClockControl : UserControl
    {
        DateTime dt;
        public ClockControl()
        {
            ResizeRedraw = true;
            Enabled = false;
```

```
            DoubleBuffered = true;
        }

        public DateTime Time
        {
            get
            {
                return dt;
            }

            set
            {
                dt = value;
                Invalidate();
            }
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics gr = e.Graphics;
            Pen pen = new Pen(ForeColor);
            Brush brush = new SolidBrush(ForeColor);
            InitializeCoordinates(gr);
            DrawDots(gr, brush);
            DrawHourHand(gr, pen);
            DrawMinuteHand(gr, pen);
            DrawSecondHand(gr, pen);
            base.OnPaint(e);
        }

        void InitializeCoordinates(Graphics gr)
        {
            if (Width == 0 || Height == 0) return;
            gr.TranslateTransform(Width / 2, Height / 2);
            float finches = Math.Min(Width , Height);
            gr.ScaleTransform(finches / 2000, finches / 2000);
        }

        void DrawDots(Graphics gr, Brush br)
        {
            for (int i = 0; i < 60; i++)
            {
                int iSize = i % 5 == 0 ? 100 : 30;
                gr.FillEllipse(br, 0 - iSize / 2, -900 - iSize / 2, iSize,
iSize);
                gr.RotateTransform(6);
            }
        }

        protected virtual void DrawHourHand(Graphics gr, Pen pen)
        {
            GraphicsState gs = gr.Save();

            int Hour;
            Hour = (Time.Hour > 12) ? Time.Hour - 12 : Time.Hour;
            gr.RotateTransform(360f / 12 * (Hour + Time.Minute / 60.0f));
        gr.DrawBeziers(pen, new Point[]
        {
         new Point(0,-600),new Point(0,-300),
         new Point(200,-300),new Point(50,-200),
         new Point(50,-200),new Point(50,0),
         new Point(50,0),new Point(50,75),
```

```
             new Point(-50,75),new Point(-50,0),
             new Point(-50,0),new Point(-50,-200),
             new Point(-50,-200),new Point(-200,-300),
             new Point(0,-300),new Point(0,-600)
            });
              gr.Restore(gs);
            }
            protected virtual void DrawMinuteHand(Graphics gr, Pen pen)
            {
                GraphicsState gs = gr.Save();
       gr.RotateTransform(360f / 60 * (Time.Minute + Time.Second / 60.0f));

    //Поворот для пользовательской системы координат (x и y меняются местами)
                gr.RotateTransform(90);
                gr.DrawPolygon(pen, new Point[]
               {
                new Point(0,-50),new Point(0,50),
                new Point(-800,0)
               });
                gr.Restore(gs);
            }

            protected virtual void DrawSecondHand(Graphics gr, Pen pen)
            {
                GraphicsState gs = gr.Save();
                gr.RotateTransform(360f / 60 * (Time.Second + Time.Millisecond /
1000.0f));
                gr.DrawLine(pen,0, 0, 0, -800);
                gr.Restore(gs);
            }
        }
    }
```

У конструкторі програма створює об’єкт класу *ClockControl*, встановлює властивість *Parent* елемента керування на форму і ініціалізує властивість *Time* поточним значенням дати й часу.

Клас *ClockControl* успадкований від класу *UserControl* і перевизначає метод *OnPaint*.

У конструкторі *ClockControl* елементу керування *ResizeRedraw* надається значення *true*, а елементу *Enabled* - *false*. Єдиному полю tm надаються значення за допомогою властивості *Time* типу *DateTime*, якому присвоєно ім’я *dt*, і доступне для читання і запису, відкрита властивість з ім’ям *Time*. Елемент керування не використовує власний таймер і не встановлює цю властивість самостійно; він лише відображає час, що відповідає значенню властивості *Time*. Оновлення значення властивості *Time* входить у завдання батьківського класу *Form1*, що створює екземпляр об’єкта *ClockControl*. За відгуком таймера в класі *Form1* через кожну секунду (*timer1.Interval = 1000*) змінюється властивість *Time* і викликається *Invalidate*.

Виклик *Invalidate* призведе до виклику методу *OnPaint*, а той у свою чергу перемалює годинник. Виклик *Invalidate* призведе до того, що фон елемента керування буде стертий і доведеться перемальовувати весь годинник цілком, що дасть в результаті подразливе мерехтіння зображення. Для усунення цього недоліку передбачено таке. *OnPaint* створює перо та пензель, використовуючи основний колір елемента керування, та викликає

п’ять інших методів. Насамперед метод *InitializeCoordinates* встановлює систему координат. Далі метод *DrawDots* малює точки, що відзначають хвилини та години. Він використовує методи класу Graphics: метод *FillEttipse* для виведення точки в позиції 12:00 та методу *RotateTransform* – для повороту на 6 градусів до наступної точки. Методи *DrawHourHand*, *DrawMinuteHand* та *DrawSecondHand* також використовують метод *RotateTransform*. При самому малюванні (годинної стрілки методом *DrawBeziers*, хвилинної стрілки методом *DrawPolygon* та секундної стрілки методом *DrawLine*). Передбачається, що стрілки спрямовано вгору. Виклик методу *RotateTransform* перед малюванням стрілок повертає їх на потрібний кут. Кожна процедура малювання стрілок містить виведення методу *Save* класу *Graphics* збереження поточного стану перед викликом *RotateTransform*. Після закінчення малювання викликається метод *Restore*.

На положення годинникової стрілки впливають дві властивості структури *DateTime*: *Hour* та *Minute*. Положення хвилинної стрілки залежить від властивостей *Minute* та *Second*, а положення секундної – від властивостей *Second* та *Millisecond*. Це дозволяє стрілкам не перескакувати з позиції на позицію, а плавно рухатись по колу. Годинникову стрілку оконтурює крива Безьє.

Тепер можна повернутись до властивості *DateTime*. Після виклику *Create.Graphics* для отримання об’єкта *Graphics* слідує виклик метода *IntiializeCoordinates*, що ініціалізує систему координат. Потім утворюється перо фонового кольору. Це потрібно для оптимального стирання стрілки, що змінює своє положення. Тут є невелика проблема: стирання будь-яку стрілку можуть пошкодити дві інші. Тому доведеться перемальовувати усі три стрілки. Незважаючи на великий обсяг малювання, такий підхід значно знижує мерехтіння зображення. У конструкторі *Form1* встановлюється властивість дочірнього елемента керування:

ClockControl. Dock = DockStyle.Fill.

Це дозволяє розгорнути годинник повністю на всю батьківську форму. Це наводиться на рис.4.32. Якщо не встановлювати властивість *Dock*, годинник буде відображено у розмірах дочого елемента. Це видно на рис. 4.33.

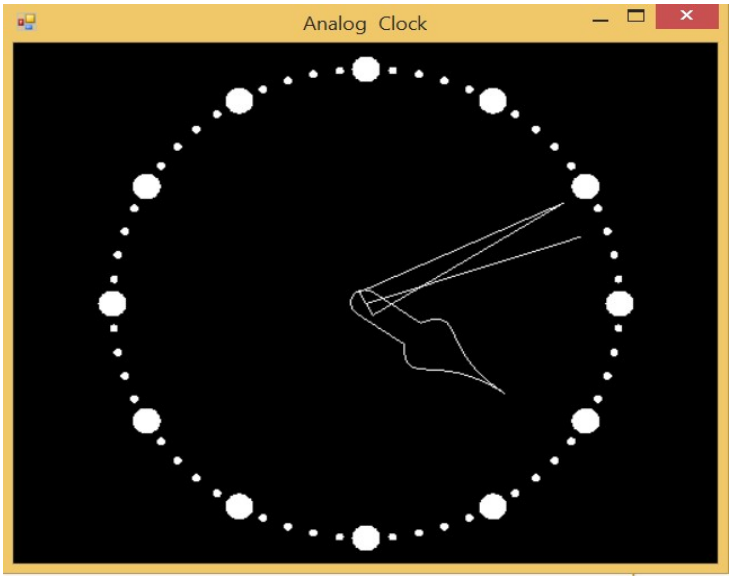


Рис. 4.32. Використання кривої Безьє та властивості *Dock*

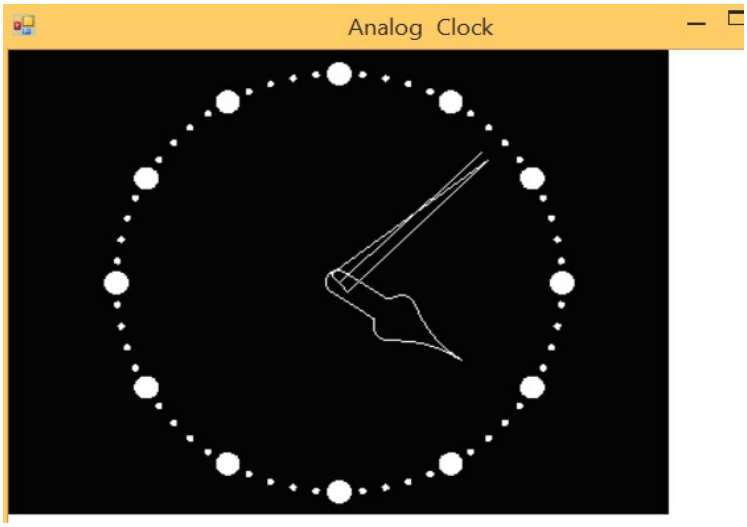


Рис. 4.33. Використання кривої Безьє

4.8. GroupBox, Panel, FlowLayoutPanel

Panel - простий елемент керування, що містить інші елементи керування. За рахунок групування разом елементів керування та приміщення їх у панель суттєво спрощується керування ними. Наприклад, можна зробити недоступними всі елементи керування на панелі, просто зробивши недоступною всю панель. Оскільки *Panel* успадковується від *ScrollableControl*, можна скористатися перевагами *AutoScroll*. Якщо в межах доступної області потрібно відобразити занадто багато елементів керування, помістіть їх у панель і встановіть значення *true* властивості *AutoScroll* - після цього їх можна буде прокручувати в межах цієї області.

Панелі за замовчуванням не відображають рамки, але, надавши значення властивості *BorderStyle*, можна візуально групувати взаємопов'язані елементи керування за допомогою рамок. Це робить інтерфейс користувача більш дружнім. *Panel* – базовий клас для *FlowLayoutPanel*, *TableLayoutPanel*, *TabPage* та *SplitterPanel*. Використовуючи ці елементи керування, можна

створювати складні та професійно виглядаючі екранні форми або вікна. *FlowLayoutPanel* і *TableLayoutPanel* особливо зручні для створення форм зі змінним розміром.

GroupBox

GroupBox є спеціальним контейнером, який відділяється від іншої форми рамкою. Він має назву, яка встановлюється за допомогою властивості *Text*. Щоб зробити *GroupBox* без заголовка, для цього у якості значення властивості *Text* просто встановлюється порожній рядок.

Нерідко цей елемент використовується для групування перемикачів – елементів *RadioButton*, оскільки дозволяє розмежувати їх групи (рис. 4.34).

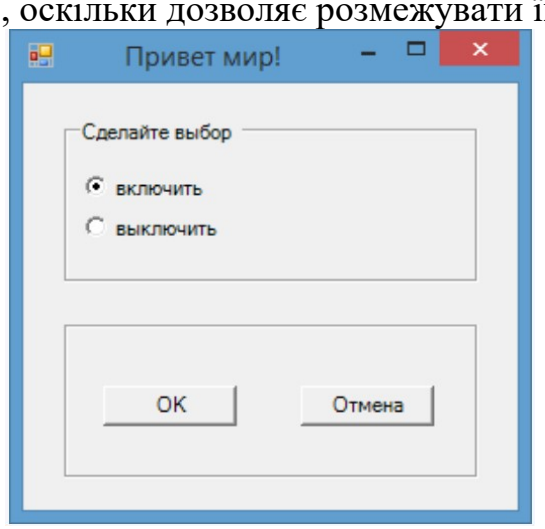


Рис. 4.34. До використання *GroupBox*

Panel

Елемент *Panel* представляє панель, а також, як і *GroupBox*, об'єднує елементи в групи. Вона може візуально зливатися з іншою формою, якщо вона має те ж значення кольору фону як *BackColor*, що і форма. Щоб її виділити, можна окрім кольору вказати для елемента рамку за допомогою властивості *BorderStyle*, яка за замовчуванням має значення *none*, тобто відсутність меж. Якщо панель має багато елементів, які виходять за її межі, ми можемо зробити панель, що прокручується, встановивши її властивість *AutoScroll* у *true* (див. рис. 4.35).

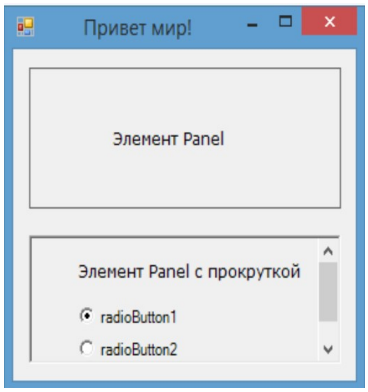


Рис. 4.35. До використання Panel.

Як і форма, *GroupBox* і *Panel* мають колекції елементів, і ми також можемо динамічно додавати в ці контейнери елементи. Наприклад, на формі є елемент *GroupBox*, який має ім'я *groupBox1*:

```
private void Form1_Load(object sender, EventArgs e)
{
        helloButton.BackColor = Color.LightGray;
        helloButton.ForeColor = Color.Red;
        helloButton.Location = new Point(30, 30);
        hel loBut on.Text = "Привет";
        groupBox1.Controls.Add(helloButton);
}
```

Для вказання розташування елемента у контейнері ми використовуємо структуру *Point: new Point(30, 30)*; котрій в конструкторі передаємо розміщення по осях *X* і *Y*. Ці координати встановлюються щодо лівого верхнього кута контейнера - тобто в даному випадку елемента *GroupBox.*

При цьому треба враховувати, що контейнером верхнього рівня є форма, а елемент *groupBox1* сам знаходиться в колекції елементів форми. І за бажання ми могли б видалити його:

```
this.Controls.Remove(groupBox1);
```

FlowLayoutPanel

Елемент *FlowLayoutPanel* успадкований від класу *Panel*, і тому успадковує всі його властивості. Однак, додаючи при цьому додаткову функціональність. Так, цей елемент дозволяє змінювати позиціонування та компонування дочірніх елементів під час зміни розмірів форми під час виконання програми.

Властивість елемента *FlowDirection* дозволяє встановити напрямок, у якому спрямовано дочірні елементи. За замовчуванням має значення *LeftToRight* - тобто елементи розташовуватимуться починаючи від лівого верхнього краю. Наступні елементи йдуть праворуч. Ця властивість також може приймати такі значення:

- *RightToLeft* – елементи розташовуються від правого верхнього кута до лівого
- *TopDown* - елементи розташовуються від верхнього лівого кута і йдуть вниз
- *BottomUp* – елементи розташовуються від лівого нижнього кута та йдуть вгору

При розміщенні елементів важливу роль відіграє властивість *WrapContents*. За замовчуванням воно має значення *true*. Це дозволяє переносити елементи, які не вміщаються у *FlowLayoutPanel*, на новий рядок або новий стовпець. Якщо воно має значення *false*, то елементи не переносяться, а до контейнера просто додаються смуги прокручування, якщо властивість *AutoScrol* дорівнює *true*.

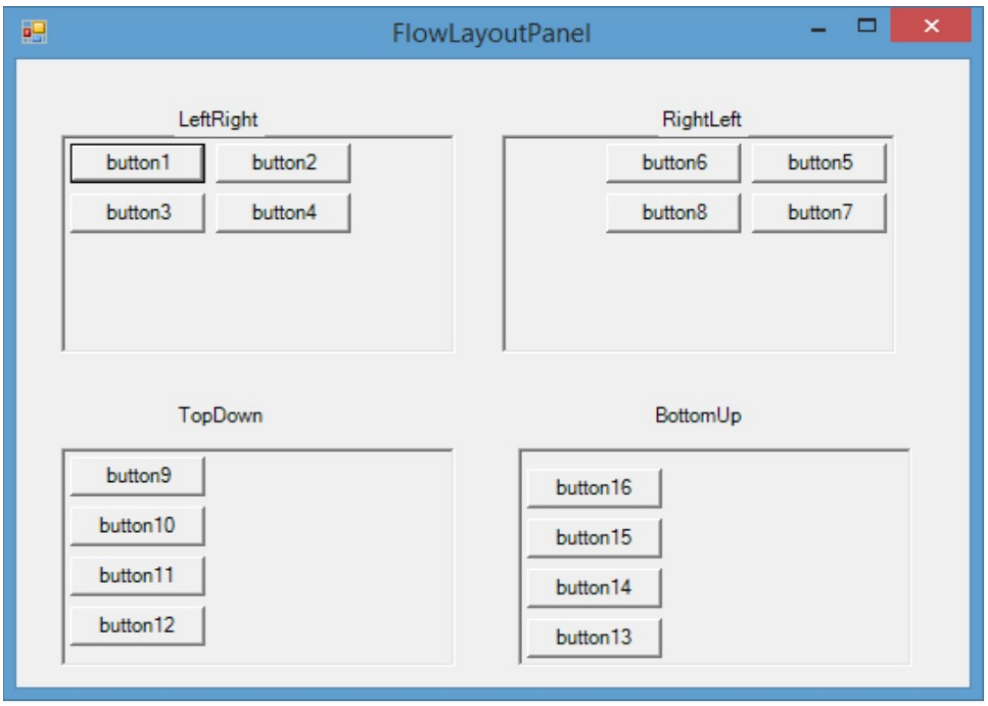


Рис. 4.36. До використання *FlowLayoutPanel*.

4.9. Головне та контекстне Меню

Для створення меню у *Windows Forms* використовується елемент *MenuStrip*. Цей клас успадкований від *ToolStrip* і тому успадковує його функціональність.

Найважливіші властивості компонента *MenuStrip*:

- *Dock*: прикріплює меню до однієї зі сторін форми
- *LayoutStyle*: вказує на орієнтацію панелі меню на формі. Може також, як і з *ToolStrip*, приймати такі значення:
    - *HorizontalStackWithOverflow*: розташування по горизонталі з переповненням - якщо довжина меню перевищує довжину контейнера, то нові елементи, що виходять за межі контейнера, не відображаються, тобто панель переповнюється елементами
    - *StackWithOverflow*: елементи розташовуються автоматично з переповненням
    - *VerticalStackWithOverflow*: елементи розташовуються вертикально з переповненням
    - *Flow*: елементи розміщуються автоматично, але без переповнення - якщо довжина панелі меню менше довжини контейнера, то елементи, що виходять за межі, переносяться
    - *Table*: елементи позиціонуються у вигляді таблиці
- *ShowItemToolTips*: вказує, чи відображатимуться підказки для окремих елементів меню.
- *Stretch*: дозволяє розтягнути панель по всій довжині контейнера
- *TextDirection*: задає напрямок тексту в пунктах меню

MenuStrip виступає свого роду контейнером окремих пунктів меню, які представлені об'єктом *ToolStripMenuItem*.

Можна виділити три способи створення меню.

1. За допомогою елемента *MenuStrip*. Додавання нових елементів меню виконується в режимі дизайнера.
2. За допомогою елемента *MenuStrip*. Додавання (видалення) елементів меню виконується у коді програми.
3. Створення та робота з меню виконується у коді програми.

Приклад 25.

Перший спосіб найпростіший. Він не вимагає програмного створення меню. Це доцільно, коли в процесі виконання програми немає потреби в зміни структури меню (створення, додавання та видалення опцій меню). Цей спосіб створення меню використовується у більшості випадків. При цьому зазвичай незначним чином коригується функція ініціалізації компонентів *InitializeComponent()*. Мається на увазі коригування відгуків опцій меню – об'єктів *ToolStripMenuItem*.

У лістингу 4.25 *menuStrip1* містить 4 об'єкти *ToolStripMenuItem* . Щоб не "плодити" функції відгуків від кожного об'єкта *ToolStripMenuItem*, доцільно всі відгуки реалізовувати в одній функції. Для цього необхідно в функціях відгуку кожного об'єкта виправити імена обробників подій на те саме ім'я. У нашому випадку, це функція *MenuItem_Click*. Реалізацію цієї функції наведено у лістингу 4.26.

Можна використати інший спосіб. Виділіть по черзі всі пункти меню, утримуючи кнопку *Ctrl*. У вікні властивостей на вкладці *Properties* натисніть двічі курсором миші по події *Click*. При цьому до всіх пунктів меню додасться обробник події *menuItemNone_Click*:

Лістинг 4.25.

```
private void InitializeComponent()
        {
            this.menuStrip1 = new System.Windows.Forms.MenuStrip();
            this.файлToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
       this.сохранитьToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
       this.удалитьToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
       this.цветToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.menuStrip1.SuspendLayout();
            this.SuspendLayout();
            //
            // menuStrip1
            //
       this.menuStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[]{
            this.файлToolStripMenuItem,
            this.цветToolStripMenuItem});
            this.menuStrip1.Location = new System.Drawing.Point(0, 0);
            this.menuStrip1.Name = "menuStrip1";
            this.menuStrip1.Size = new System.Drawing.Size(282, 28);
            this.menuStrip1.TabIndex = 0;
            this.menuStrip1.Text = "menuStrip1";
```

```
                //
                // файлToolStripMenuItem
                //
                this.файлToolStripMenuItem.DropDownItems.AddRange(new
                System.Windows.Forms.ToolStripItem[] {
                this.сохранитьToolStripMenuItem,
                this.удалитьToolStripMenuItem});
                this.файлToolStripMenuItem.Name = "файлToolStripMenuItem";
                this.файлToolStripMenuItem.Size = new System.Drawing.Size(61, 24);
                this.файлToolStripMenuItem.Text = " Файл";
                //
                // сохранитьToolStripMenuItem
                //
                this.сохранитьToolStripMenuItem.Checked = true;
                this.сохранитьToolStripMenuItem.CheckState =
                System.Windows.Forms.CheckState.Checked;
                this.сохранитьToolStripMenuItem.Name =
"сохранитьToolStripMenuItem";
                this.сохранитьToolStripMenuItem.Size = new System.Drawing.Size(152,
24);
                this.сохранитьToolStripMenuItem.Text = "Сохранить";
                this.сохранитьToolStripMenuItem.Click += new
                System.EventHandler(this.MenuItem_Click);
                //
                // удалитьToolStripMenuItem
                //
                this.удалитьToolStripMenuItem.Name = "удалитьToolStripMenuItem";
                this.удалитьToolStripMenuItem.Size=new System.Drawing.Size(152,24);
                this.удалитьToolStripMenuItem.Text = "Удалить";
                this.удалитьToolStripMenuItem.Click += new
                System.EventHandler(this.MenuItem_Click);
                //
                // цветToolStripMenuItem
                //
                this.цветToolStripMenuItem.Checked = true;
                this.цветToolStripMenuItem.CheckState =
                System.Windows.Forms.CheckState.Checked;
                this.цветToolStripMenuItem.Name = "цветToolStripMenuItem";
                this.цветToolStripMenuItem.Size = new System.Drawing.Size(54, 24);
                this.цветToolStripMenuItem.Text = "Цвет";
                this.цветToolStripMenuItem.Click += new
                System.EventHandler(this.MenuItem_Click);
                //
                // Form1
                //
                this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
                this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
                this.ClientSize = new System.Drawing.Size(282, 253);
                this.Controls.Add(this.menuStrip1);
                this.MainMenuStrip = this.menuStrip1;
                this.Name = "Form1";
                this.Text = "Form1";
                this.menuStrip1.ResumeLayout(false);
                this.menuStrip1.PerformLayout();
                this.ResumeLayout(false);
                this.PerformLayout();
            }
            #endregion
            private System.Windows.Forms.MenuStrip menuStrip1;
            private System.Windows.Forms.ToolStripMenuItem файлToolStripMenuItem;
            private System.Windows.Forms.ToolStripMenuItem
сохранитьToolStripMenuItem;
```

```
            private System.Windows.Forms.ToolStripMenuItem
удалитьToolStripMenuItem;
            private System.Windows.Forms.ToolStripMenuItem цветToolStripMenuItem;
```

Листинг 4.26.

```
namespace examp25
{
    public partial class Form1 : Form
    {
        public Form1() {
            InitializeComponent();
        }
        private void MenuItem_Click(object sender, EventArgs e)
            Text = sender.ToString();
        }
    }
}
```

На рис 4.37. наведено фрагмент програми реалізації меню першим способом.

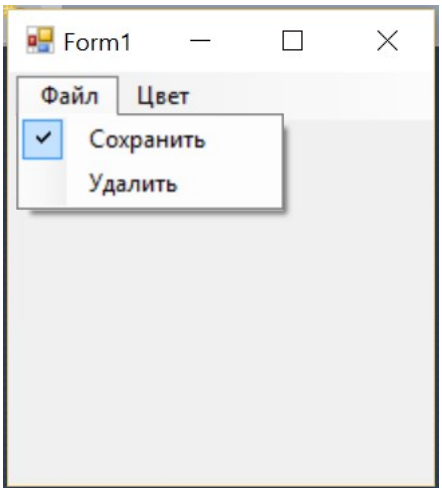


Рис. 4.37. До реалізації меню 1 способом

Приклад 26.

Другий спосіб створення меню ґрунтується на використанні елемента керування *menuStrip*. При цьому формування меню виконується у програмному режимі. Цей спосіб доцільно використовувати, коли у процесі роботи програми необхідне створення, доповнення та видалення опцій меню. На лістингу 4.27. та рис 4.38. наведено лістинг реалізації меню та його вигляд.

Лістинг 4.27. (examp26)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
namespace examp26
{
    public partial class Form1 : Form
    {
        public Form1()
        {
        InitializeComponent();
        ToolStripMenuItem fileItem = new ToolStripMenuItem("Файл");
        fileItem.DropDownItems.Add("Создать");
        fileItem.DropDownItems.Add(new ToolStripMenuItem("Сохранить"));
        menuStrip1.Items.Add(fileItem);
    ToolStripMenuItem aboutItem = new ToolStripMenuItem("О программе");
        aboutItem.Click += aboutItem_Click;
        menuStrip1.Items.Add(aboutItem);
    }

    void aboutItem_Click(object sender, EventArgs e)
    {
        MessageBox.Show("О программе");
    }
  }
}
```

ToolStripMenuItem у конструкторі приймає текстову мітку, яка використовуватиметься як текст меню. Кожен такий об'єкт має колекцію *DropDownItems*, яка зберігає дочірні об'єкти *ToolStripMenuItem*. Тобто, один елемент *ToolStripMenuItem* може містити набір інших об'єктів *ToolStripMenuItem*. І таким чином утворюється ієрархічне меню або структура у вигляді дерева.

Якщо передати при додаванні рядок тексту, то для неї неявно буде створено об'єкт *ToolStripMenuItem*: *fileItem.DropDownItems.Add("Створити")*

Призначивши обробники для події *Click*, ми можемо обробити натискання пунктів меню: *aboutItem.Click += aboutItem_Click.*

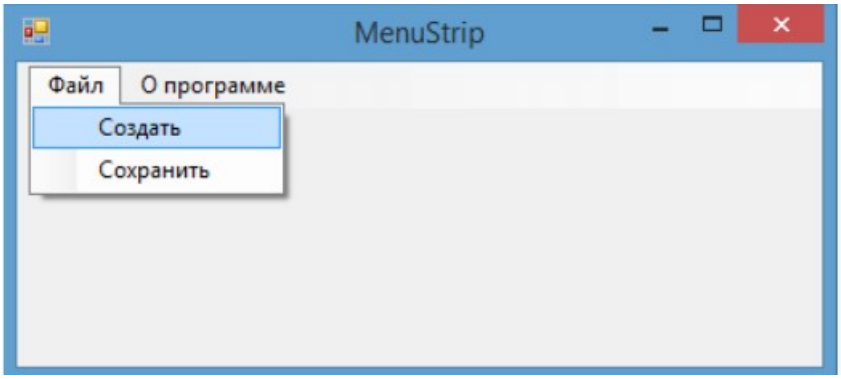


Рис. 4.38. До реалізації 2 способом

Приклад 27.

Під час створення меню надається можливість позначити пункти меню. Це виконується за допомогою властивостей *CheckOnClick*, *Checked* та *CheckState*

Властивість *CheckOnClick* при значенні *true* дозволяє відобразити пункт меню. А за допомогою властивості *Checked* можна встановити, чи буде пункт

меню відзначений під час запуску програми. Властивість *CheckState* повертає стан пункту меню – відзначений він чи ні. Воно може приймати три значення: *Checked* (позначений), *Unchecked* (непомічений) та *Indeterminate* (у невизначеному стані)

На лістингу 4.28. наведено код програмування позначок опцій меню. Вигляд такого меню наведено на рис. 4.39.

Лістинг 4.28. (examp27)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp27
{
    public partial class Form1 : Form
    {
        public Form1()
        {
          InitializeComponent();
          ToolStripMenuItem fileItem = new ToolStripMenuItem("Файл");
          ToolStripMenuItem newItem = new ToolStripMenuItem("Создать")
          { Checked = true, CheckOnClick = true };
        fileItem.DropDownItems.Add(newItem);
        ToolStripMenuItem saveItem = new ToolStripMenuItem("Сохранить")
        { Checked = true, CheckOnClick = true };
        saveItem.CheckedChanged += menuItem_CheckedChanged;
        fileItem.DropDownItems.Add(saveItem);
        menuStrip1.Items.Add(fileItem);
    }
        void menuItem_CheckedChanged(object sender, EventArgs e)
    {
        ToolStripMenuItem menuItem = sender as ToolStripMenuItem;
        if (menuItem.CheckState == CheckState.Checked)
        MessageBox.Show("Отмечен");
        else if (menuItem.CheckState == CheckState.Unchecked)
        MessageBox.Show("Отметка снята");
    }
}
```

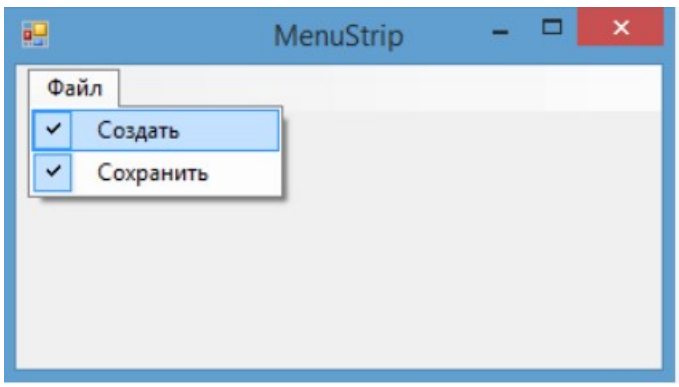


Рис. 4.39. До розробки меню з відмітками їх опцій

Приклад 28.

Під час створення меню можна використовувати клавіші швидкого доступу. Для призначення клавіш швидкого доступу використовується властивість *ToolStripMenuItem.ShortcutKeys*. Клавіші задаються за допомогою *Keys*. У лістингу 4.29 наведено програму, в якій після натискання на комбінацію клавіш *Ctrl + P* буде спрацьовувати натискання по пункту меню "Зберегти".

Лістинг 4.29. (examp28)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp28
{

public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        ToolStripMenuItem fileItem = new ToolStripMenuItem("Файл");
        ToolStripMenuItem saveItem = new ToolStripMenuItem("Сохранить")
        {Checked = true, CheckOnClick = true };
        saveItem.Click+=saveItem_Click;
         saveItem.ShortcutKeys = Keys.Control | Keys.P;
         fileItem.DropDownItems.Add(saveItem);
         menuStrip1.Items.Add(fileItem);
    }

    void saveItem_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Сохранение");
    }
}
```

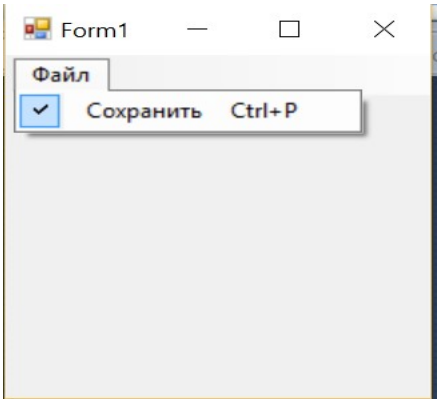


Рис. 4.40. До використання клавіш швидкого доступу

Приклад 29.

Третій спосіб створення меню використовувався у старих версіях, коли не було дизайнера та програмування (у тому числі робота з меню) виконувалося повністю в ручному режимі. У лістингу 4.30. та рис. 4.41. наводиться реалізація створення меню.

Лістинг 4.30. (examp29)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace examp29
{
    //Работа с меню
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            Text = "First Main Menu";
             MenuItem M_Open, M_Save;
             M_Open = new MenuItem("&Open...",
             new EventHandler(FileOpen),
             Shortcut.CtrlO);
             M_Save = new MenuItem("&Save...",
             new EventHandler(FileSave),Shortcut.CtrlS);
            MenuItem mFile = new MenuItem("&File",
            new MenuItem[] { M_Open, M_Save });
            Menu = new MainMenu(new MenuItem[] { mFile });
         }
        void FileOpen(object obj, EventArgs ea)
        {
            MessageBox.Show("Open");
        }
        void FileSave(object obj, EventArgs ea)
        {
            MessageBox.Show("Save");
        }
    }
}
```

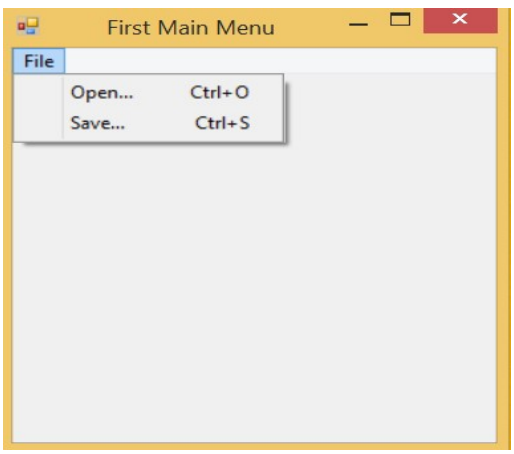


Рис. 4.41. До розробки меню 3 способом (без використання дизайнера)

Ми розглянули 3 способи створення меню. Додатково варто зазначити ще одну можливість при створенні меню. Для додавання опцій меню можна використовувати три види елементів: *MenuItem* (об'єкт *ToolStripMenuItem*), *ComboBox* та *TextBox*. Таким чином, у меню ми можемо використовувати списки, що спадають, і текстові поля, однак, як правило, ці елементи застосовуються переважно на панелі інструментів. Меню зазвичай містить набір об'єктів *ToolStripMenuItem*. Вигляд такого меню наводиться на рис. 4.42.

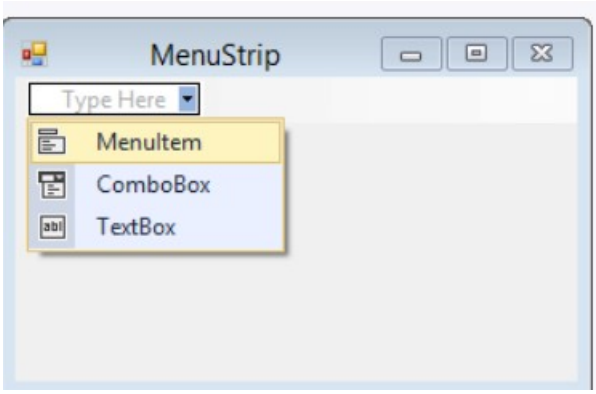


Рис. 4.42. Створення меню з використанням 3 видів елементів керування

Контекстне меню

Клас *ContextMenuStrip* представляє контекстне меню. Даний компонент багато в чому аналогічний елементу *MenuStrip* за винятком, що контекстне меню не може використовуватися саме по собі, воно обов'язково застосовується до якогось іншого елемента, наприклад, текстового поля або форми. Нові елементи в контекстному меню можна додати в режимі дизайнера. При цьому ми можемо додати ті самі елементи, що і в *MenuStrip*. Як правило, використовується *ToolStripMenuItem* або елемент *ToolStripSeparator*, що представляє горизонтальну смужку - роздільник між іншими пунктами меню.

Тепер створимо невелику програму. Додамо на форму елементи *ContextMenuStrip* та *TextBox*, які матимуть назви *contextMenuStrip1* та *textBox1* відповідно. Потім змінимо код форми наступним чином:

```
public partial class Form1 : Form
{
    string buffer;
    public Form1()
    {
        InitializeComponent();
        textBox1.Multiline = true;
        textBox1.Dock = DockStyle.Fill;
        // создаем элементы меню
        ToolStripMenuItem copyMenuItem = new ToolStrip
         MenuItem("Копировать");
        ToolStripMenuItem pasteMenuItem = new ToolStripMenuItem("Вставить");
        // добавляем элементы в меню
        contextMenuStrip1.Items.AddRange(new[]{ copyMenuItem, pasteMenuItem });
        // ассоциируем контекстное меню с текстовым полем
        textBox1.ContextMenuStrip = contextMenuStrip1;
```

```
            // устанавливаем обработчики событий для меню
            copyMenuItem.Click += copyMenuItem_Click;
            pasteMenuItem.Click += pasteMenuItem_Click;
        }
        // вставка текста
        void pasteMenuItem_Click(object sender, EventArgs e)
        {
            textBox1.Paste(buffer);
        }
        // копирование текста
        void copyMenuItem_Click(object sender, EventArgs e)
        {
        // если выделен текст в текстовом поле, то копируем его в буфер
            buffer = textBox1.SelectedText;
        }
}
```

У даному випадку виконано найпростішу реалізацію функціональності *copy-paste*. У меню додається два елементи. А в текстові поля встановлюється багаторядковість і воно розтягується по ширині контейнера (рис. 4.43).

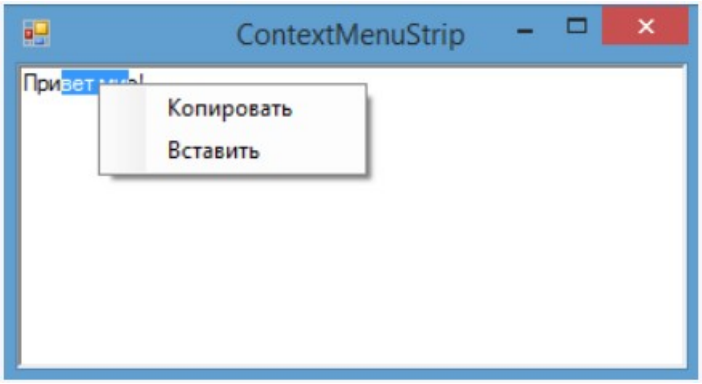


Рис. 4.43. Контекстне меню з прив’язкою (асоціацією) до TextBox

Багато компонентів мають властивість *ContextMenuStrip*, яка дозволяє асоціювати контекстне меню з даним елементом. У випадку з *TextBox,* асоціація відбувається так: *textBox1.ContextMenuStrip = contextMenuStrip1*. І після натискання на текстове поле правою кнопкою миші ми зможемо викликати асоційоване контекстне меню.

За допомогою обробників натискання пунктів меню встановлюються дії копіювання та вставки рядків.

Приклад 30.

Розглянемо приклад створення контекстного меню із прив’язкою до форми. У цьому прикладі використовується мінімальний код. У режимі дизайнера встановлюється компонент *ContextMenuStrip*. Після цього до властивості форми *ContextMenuStrip* присвоюється значення *contexMenuStrip1*, як це показано на рис. 4.44.

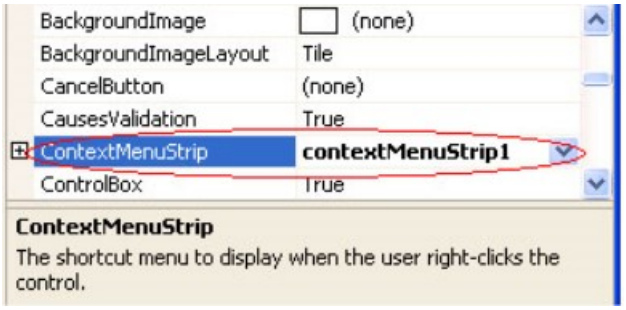


Рис. 4.44. До асоціації контекстного меню до форми

У режимі дизайнера при натисканні на опції меню автоматично формуються відгуки функції *InitializeComponent()*. Текст програми та вид контекстного меню, що з'являється на формі при натисканні правої кнопки миші, показано відповідно на лістингу 4.31. та рис. 4.45.

Лістинг 4.31. (examp30)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;


namespace exam12
{

    // Работа с контекстным меню
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

private void a1ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show("A1");
        }

private void a2ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show("A2");
        }

private void a3ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show("A3");
        }
    }
}
```

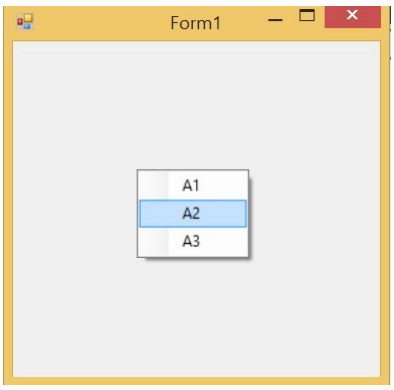


Рис. 4.45. Контекстне меню з прив’язкою (асоціацією) до Form

4.10. Стандартні діалоги

Перш ніж розглядати стандартні діалоги, нагадаємо створення діалогів у вигляді форм, розглянутих у розділі 3.1. При створенні складних додатків дуже часто основні дані визначено і їх обробка виконується в основній формі. Для ініціалізації чи зміни даних використовуються діалоги як окремі форми.

Приклад 31.

На лістингу 4.32. та рис. 4.46. наведено спрощений приклад, що складається із двох форм: основної форми (*Form1*), яка викликає діалог - форму (*Form2*).

Лістинг 4.32. (examp31)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace exam31
{
    //Работа с диалогом
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void dialogToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Form2 dlg = new Form2();
            dlg.ShowDialog();

            int f = (int)dlg.DialogResult +100;
            MessageBox.Show(f.ToString());
       }
    }
}
```

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace exam31
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
         this.DialogResult = (DialogResult)1;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.Cancel;
        }
    }
}
```

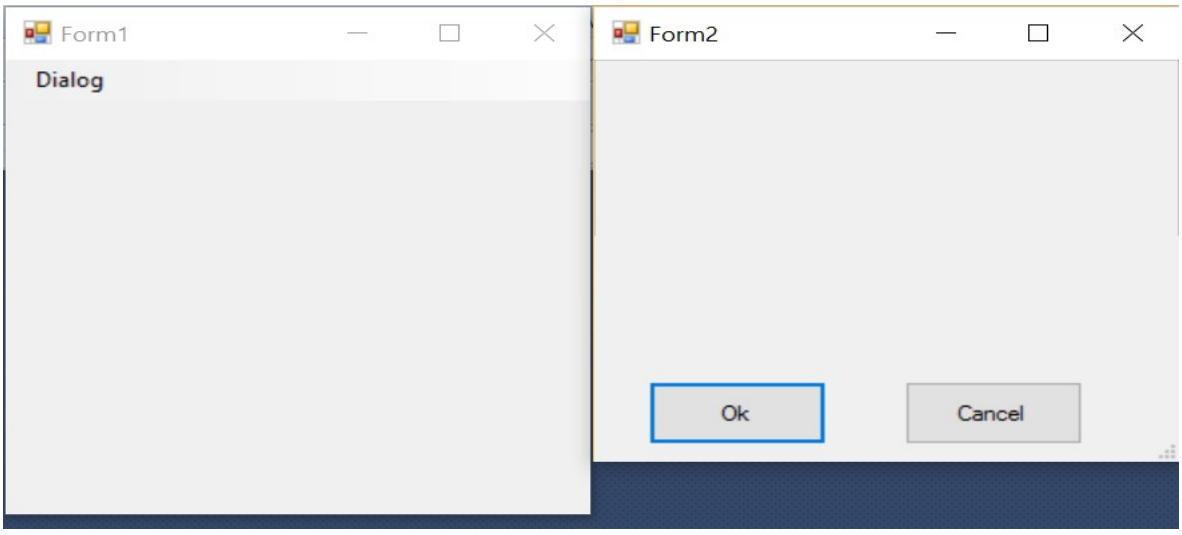


Рис. 4.46. До створення діалога

Розглянемо стандартні діалоги вибору шрифту, відкриття та збереження файлу.

Для вибору шрифту та його параметрів використовується *FontDialog*. Для його використання перенесемо компонент із панелі інструментів на форму. Також, нехай на формі є кнопка *button1*. Тоді код форми має вигляд так:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        button1.Click += button1_Click;
        // добавляем возможность выбора цвета шрифта
        fontDialog1.ShowColor = true;
    }

    void button1_Click(object sender, EventArgs e)
    {
        if (fontDialog1.ShowDialog() == DialogResult.Cancel)
            return;

        // установка шрифта
        button1.Font = fontDialog1.Font;
        // установка цвета шрифта
        button1.ForeColor = fontDialog1.Color;
    }
}
```

FontDialog має низку властивостей, серед яких варто відзначити такі:

- *ShowColor*: під час значення *true* дозволяє вибирати колір шрифту;
- *Font*: вибраний у діалоговому вікні шрифт;
- *Color*: вибраний у діалоговому вікні колір шрифту.

Для відображення діалогового вікна використовується метод *ShowDialog()*.

Якщо ми запустимо програму і натиснемо на кнопку, то відобразиться діалогове вікно (рис. 4.47), де ми можемо задати всі параметри шрифту. І після вибору встановлені налаштування будуть застосовані до шрифту кнопки:

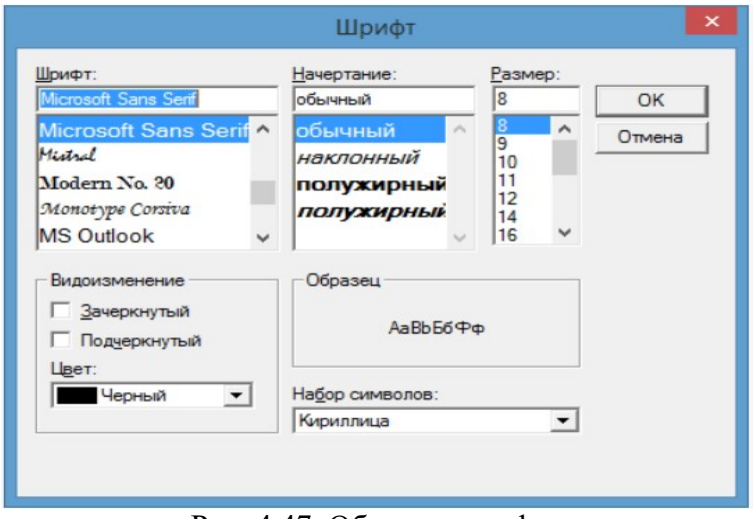


Рис. 4.47. Обрання шрифту

ColorDialog дозволяє вибрати параметри кольору. Також перенесемо його із панелі інструментів на форму. І змінимо код форми:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        button1.Click += button1_Click;
        // расширенное окно для выбора цвета
        colorDialog1.FullOpen = true;
        // установка начального цвета для colorDialog
        colorDialog1.Color = this.BackColor;
    }

    void button1_Click(object sender, EventArgs e)
    {
        if (colorDialog1.ShowDialog() == DialogResult.Cancel)
            return;
        // установка цвета формы
        this.BackColor = colorDialog1.Color;
    }
}
```

Серед властивостей *ColorDialog* слід зазначити такі:

- *FullOpen*: при значенні *true* відображається діалогове вікно з розширеними налаштуваннями для вибору кольору;
- *SolidColorOnly*: при значенні *true* дозволяє вибирати тільки між однотонними відтінками кольорів;
- *Color*: вибраний у діалоговому вікні колір.

І при натисканні кнопки з’явиться діалогове вікно, в якому встановлюється колір форми. Таке вікно наведено на рис. 4.48.

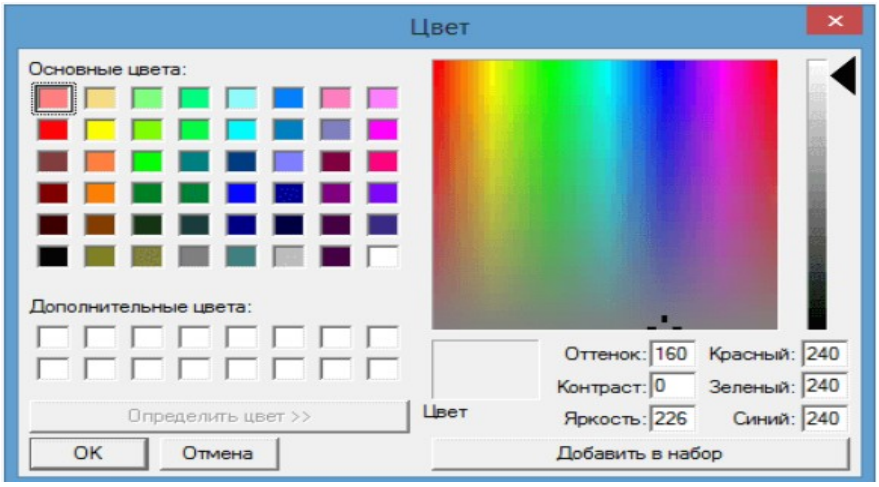


Рис. 4.48. Обрання кольору

Вікна відкриття та збереження файлу представлені класами *OpenFileDialog* та *SaveFileDialog*. Вони мають багато в чому схожу функціональність, тому розглянемо їх разом. *OpenFileDialog* та *SaveFileDialog* мають ряд загальних властивостей, серед яких можна виділити такі:

- *DefaultExt*: встановлює розширення файлу, яке додається за замовчуванням, якщо користувач запровадив ім’я файлу без розширення
- *AddExtension*: при значенні *true* додає до імені файлу розширення за його відсутності. Розширення береться з властивості *DefaultExt* або *Filter*
- *CheckFileExists*: якщо має значення *true*, перевіряє існування файлу з вказаним ім’ям
- *CheckPathExists*: якщо має значення *true*, то перевіряє існування шляху до файлу з зазначеним ім’ям
- *FileName*: повертає повне ім’я файлу, обраного у діалоговому вікні
- *Filter*: задає фільтр файлів, завдяки чому у діалоговому вікні можна відфільтрувати файли за розширенням. Фільтр задається у наступному форматі *назва_файлів|*.розширення*. Наприклад, текстові файли *(*.txt)|*.txt*. Можна здати відразу кілька фільтрів, при цьому вони поділяються вертикальної лінією |. Наприклад, *Bitmap files (*.bmp)|*.bmp|Image files (*.jpg)|*.jpg*
- *InitialDirectory*: встановлює каталог, який відображається під час виклику вікна
- *Title*: заголовок діалогового вікна
- Окремо у класу *SaveFileDialog* можна виділити кілька властивостей:
- *CreatePrompt*: при значенні *true* у випадку, якщо вказано не існуючий файл, відображатиметься повідомлення про його створення
- *OverwritePrompt*: при значенні *true* у випадку, якщо вказаний існуючий файл, відображатиметься повідомлення про те, що файл буде перезаписано

Щоб відобразити діалогове вікно, необхідно викликати *ShowDialog()*.

Розглянемо обидва діалогові вікна на прикладі. Додамо на форму текстове поле *textBox1* та дві кнопки *button1* та *button2*. Також перетягнемо з панелі інструментів компоненти *OpenFileDialog* та *SaveFileDialog*. Після додавання вони з’являться знизу дизайнера форми. У результаті форма виглядатиме приблизно так, як показано на рис. 4.49.

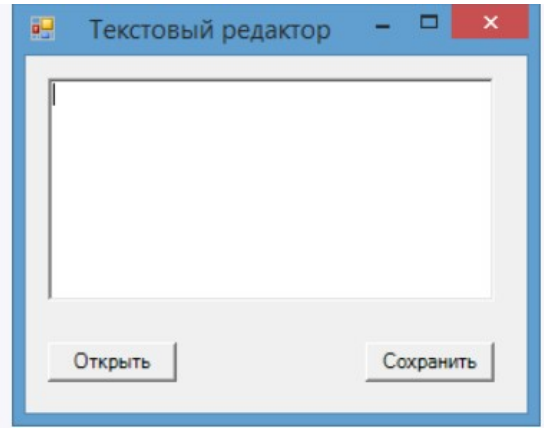


Рис. 4.49. Робота з файлами

Код форми:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        button1.Click += button1_Click;
        button2.Click += button2_Click;
        openFileDialog1.Filter = "Text files(*.txt)|*.txt|All files(*.*)|*.*";
        saveFileDialog1.Filter = "Text files(*.txt)|*.txt|All files(*.*)|*.*";
    }
    // сохранение файла
    void button2_Click(object sender, EventArgs e)
    {
        if (saveFileDialog1.ShowDialog() == DialogResult.Cancel)
            return;
        // получаем выбранный файл
        string filename = saveFileDialog1.FileName;
        // сохраняем текст в файл
        System.IO.File.WriteAllText(filename, textBox1.Text);
        MessageBox.Show("Файл сохранен");
    }
    // открытие файла
    void button1_Click(object sender, EventArgs e)
    {
        if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
            return;

        // получаем выбранный файл
        string filename = openFileDialog1.FileName;
        // читаем файл в строку
        string fileText = System.IO.File.ReadAllText(filename);
        textBox1.Text = fileText;
        MessageBox.Show("Файл открыт");
    }
}
```

Приклад 32.

Розглянемо приклад використання *FontDialog*, *ColorDialog* та *SaveFileDialog*. Як видно на рис. 4.50., вихідними елементами керування є 3 кнопки для підключення відповідно 3 діалогів. У *Label* наводяться результати. Змінюється шрифт тексту *Hello*, а також повний шлях збереження файлу *Help*. Текст програми наводиться на лістингу 4.33.

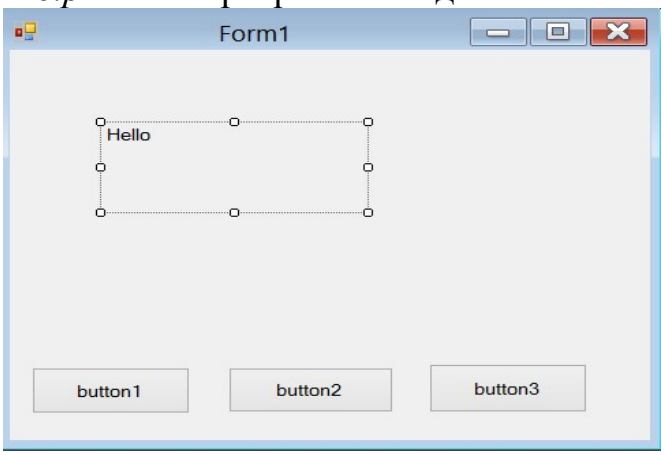


Рис. 4.50. Робота зі стандартними діалогами

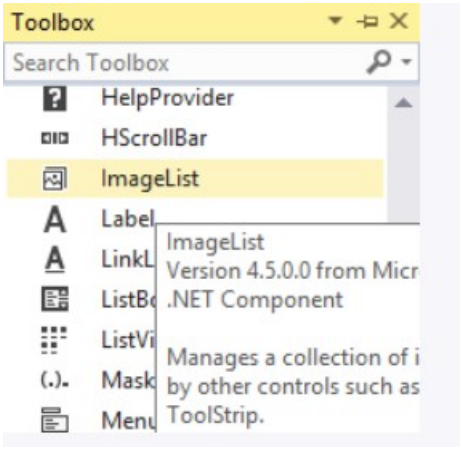
Лістинг 4.33. (examp32)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp32
{
    // Работа со стандартными диалогами

    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            fontDialog1.Font = lbl.Font;
            if (fontDialog1.ShowDialog() == DialogResult.OK)
            lbl.Font = fontDialog1.Font;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            colorDialog1.Color = lbl.ForeColor;
            if(colorDialog1.ShowDialog()==DialogResult.OK)
            lbl.ForeColor = colorDialog1.Color;
        }

        private void button3_Click(object sender, EventArgs e)
        {
            saveFileDialog1.FileName = lbl.Text;
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
                lbl.Text = saveFileDialog1.FileName;
        }
    }
}
```

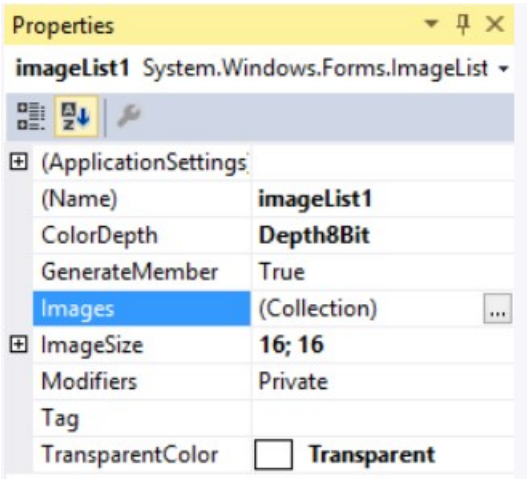
4.11. Елемент керування ImageList

Об’єкт *ImageList* містить точкові малюнки (набір зображень), які використовуються в різних елементах керування. Цей об’єкт переважно використовується в *ListView*, *TreeView*, *ToolBar*.

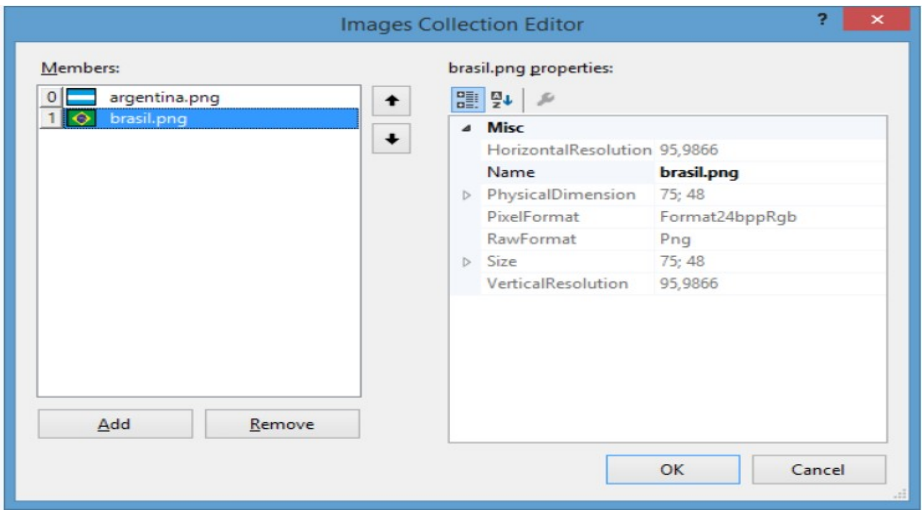
ImageList переноситься на форму з панелі інструментів:



Оскільки компонент *ImageList* не є візуальним елементом, ми побачимо його під формою. Ключовою властивістю *ImageList* є властивість *Images*, яка визначає колекцію зображень.



При виборі цієї властивості нам відкриється вікно редактора зображень, у якому ми можемо додати нове зображення або видалити наявне.



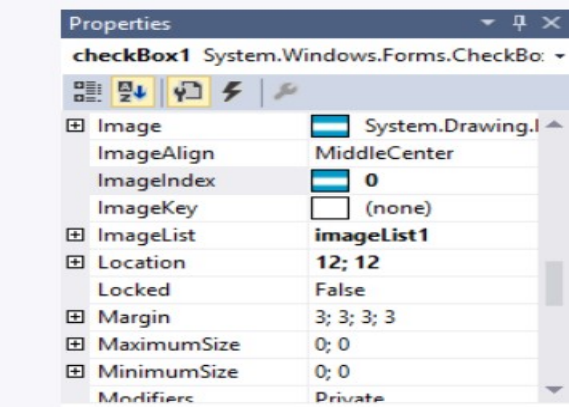
Для встановлення розміру зображень для цього *ImageList* можна використовувати його властивість *ImageSize*. За замовчуванням ширина та

висота мають значення 16 пікселів, але ми можемо встановити будь-яке інше, але не більше 256 пікселів.

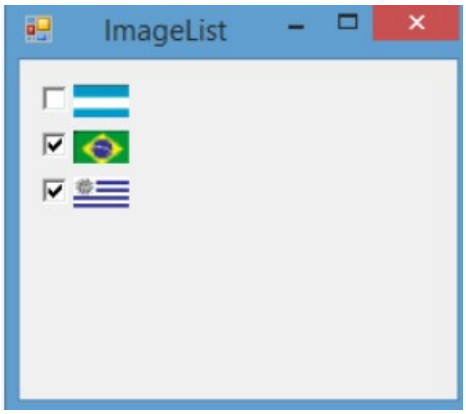
Також можна додавати/вилучати зображення зі списку програмно:

```
ImageList1.Images.Add(Image.FromFile(@"C:\Users\Pictures\
uruguay.png"));
imageList1.Images.RemoveAt(0); // удаляем первое изображение
```

Для прикладу додамо в *ImageList* три зображення і помістимо на форму три чекбокси. У кожного чекбокса приберемо тест і встановимо властивість **ImageList** і вкажемо у властивості *ImageIndex* індекс зображення з *imageList1*:



Отримаємо форму на зразок наступної:



Слід додати, що для створення іконки підходить навіть редактор *Paint*. Для іконки панелі інструментів розмір малюнка має бути 16x16 пікселів, збережений як 256 кольоровий малюнок. Для перетворення збереженого малюнка, наприклад 1.bmp, просто змініть його розширення на 1.ico.

4.12. Панель інструментів (елементи керування ToolBar, ToolStrip)

Для створення панелі інструментів використовуються елементи керування *TooBar* та *ToolScrip*, який більш вдосконалений та має місце в останніх версіях *C#*.

Однак багато програм використовують *ToolBar* і багато програмістів влаштовують його можливості. Тому доцільно розглянути *TooBar* та *ToolScrip*.

Елемент керування ToolBar

Об'єкт *ToolBar* (панель інструментів) є контейнером для групи команд або елементів керування, які зазвичай пов'язані між собою за функціональністю. Панель інструментів зазвичай містить кнопки, які викликають команди. Ці кнопки представлені картинками, тому створення *ToolBar* безпосередньо пов'язано з додаванням до *ImageList* бітових зображень через його властивість *Images*, як це було показано вище, можна виконати це додавання на основі роботи з ресурсами. У *C#* здебільшого як ресурси використовуються бітові образи: картинки, іконки. Для входу до редактора ресурсів необхідно увійти в *Properties*. Далі можна вибрати ресурс із файлу або створити новий. У будь-якому випадку ці файли будуть розміщені в папці *Resources*. При виборі файлу ззовні - він буде скопійований туди. Ім'я ресурсу буде таке саме як і ім'я файлу але без розширення. Для того, щоб завантажити ресурс у коді програми, необхідно прописати *Properties.Resources.<ім'я ресурсу>*. Тому, якщо ресурс називається *Image1*, то для його завантаження буде рядок *Properties.Resources.Image1*. Розглянемо два приклади створення *ToolBar*.

Приклад 33.

ImageList переноситься на форму з панелі інструментів. *ToolBar* створюється у програмі. Як ресурси використано три файли з розширенням *.bmp* та іконки *.ICO*. Це видно на малюнку 4.51. На рис. 4.52 показано реалізацію програмування меню та панелі інструментів.

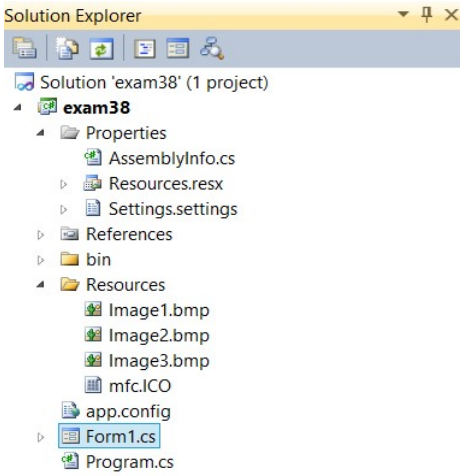


Рис. 4.51. Solution Explorer

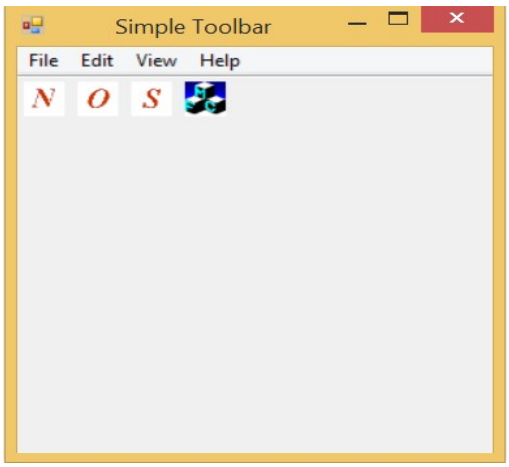


Рис. 4.52. До використання ToolBar.

Реалізацію програми наведено у лістингу 4.34.

Лістинг 4.34. (examp33)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp33
{
    //Работа с меню и панелью инструментов (класс ToolBar)

    public partial class Form1 : Form
    {
        ToolBar tbar;
        public Form1()
        {
            InitializeComponent();
            Text = "Simple Toolbar";
            Menu = new MainMenu();
            Menu.MenuItems.Add( "File");
            Menu.MenuItems.Add("Edit");
            Menu.MenuItems.Add ("View");
            Menu.MenuItems.Add("Help");
            Menu.MenuItems[0].Click += new EventHandler(MenuOnClick);

            // Создание набора изображения
            // Загрузка картинок из ресурсов

             this.imageList1.Images.Clear();
             this.imageList1.Images.Add(Properties.Resources.Image1);
             this.imageList1.Images.Add(Properties.Resources.Image2);
             this.imageList1.Images.Add(Properties.Resources.Image3);
             this.imageList1.Images.Add(Properties.Resources.mfc);

            // Создание панели инструментов
            tbar = new ToolBar();
            tbar.Parent = this;
            tbar.ImageList = imageList1;
            tbar.ShowToolTips = true;
```

```
                string[] astr = {"New", "Open", "Save","MFC"};

                for (int i=0 ; i < imageList1.Images.Count; i++)
                {
                   ToolBarButton tbarbtn = new ToolBarButton();
                   tbarbtn.ImageIndex = i;
                   tbarbtn.ToolTipText = astr[i];
                   tbar.Buttons.Add(tbarbtn);
                }
                // Отклик на ToolBar
                tbar.ButtonClick += new
ToolBarButtonClickEventHandler(ToolBarClick);
            }
            //Событие на меню
            void MenuOnClick(object obj, EventArgs ea)
            {
              MessageBox.Show(((MenuItem)obj).Text);
            }

            // Событие на toolbar
            void ToolBarClick(object obj, ToolBarButtonClickEventArgs ea)
            {
                //Действия на индекс кнопки
                switch (tbar.Buttons.IndexOf(ea.Button))
                {
                    case 0:
                        MessageBox.Show("New");
                        break;
                    case 1:
                        MessageBox.Show("Open");
                        break;
                    case 2:
                        MessageBox.Show("Save");
                        break;
                    case 3:
                        MessageBox.Show("MFC");
                        break;
                }
            }
        }
    }
```

Приклад 34.

Тут мають місце такі відмінності. Наведено *ToolBar* без *Menu. ImageList* та *ToolBox* переносяться на форму з панелі інструментів. Зображення *Image1*, *Image2*, *Image3* завантажуються з ресурсів. Іконка завантажується з файлу методом *FromFile*. Результати програмування наведено на лісингу 4.35 та рис. 4.53.

Лістинг 4.35. (examp34)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace examp34 {
```

```
    //Работа с панелью инструментов (класс ToolStrip)
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            Text = "Simple Toolbar";
            this.imageList1.Images.Clear();
            this.imageList1.Images.Add(Properties.Resources.Image1);
            this.imageList1.Images.Add(Properties.Resources.Image2);
            this.imageList1.Images.Add(Properties.Resources.Image3);
            this.imageList1.Images.Add(Image.FromFile(@"D:\mfc.ICO"));
            // Работа с панелью инструментов
            toolStrip1.ImageList = imageList1;
            toolStrip1.ShowItemToolTips = true;
            string[] astr = {"New", "Open", "Save","MFC"};

            for (int i=0 ; i < imageList1.Images.Count; i++)
            {
               ToolStripButton tbarbtn = new ToolStripButton();
               tbarbtn.ImageIndex = i;
               tbarbtn.ToolTipText = astr[i];
               toolStrip1.Items.Add(tbarbtn);
            }
        }
        // Событие на ToolStrip
        private void toolStrip1_ItemClicked(object sender,
        ToolStripItemClickedEventArgs e)
        {
            //Действия на индекс кнопки
            switch (toolStrip1.Items.IndexOf(e.ClickedItem))
            {
                case 0:
                    MessageBox.Show("New");
                    break;
                case 1:
                    MessageBox.Show("Open");
                    break;
                case 2:
                    MessageBox.Show("Save");
                    break;
            }
        }
    }
}
```

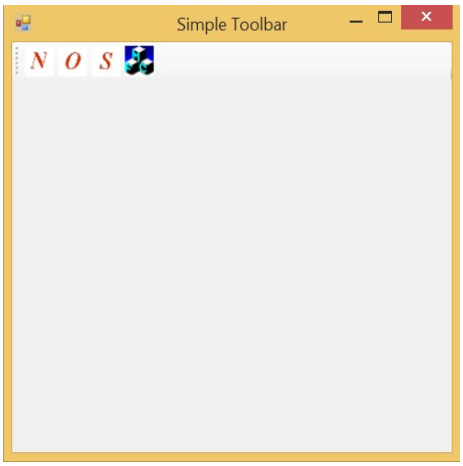


Рис. 4.53. Панель інструментів

Елемент керування *ToolStrip*

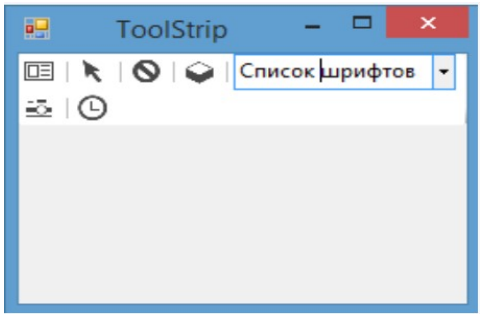
Елемент керування *ToolStrip* - це контейнер, який використовується для створення панелей інструментів, структур меню та рядків стану. *ToolStrip* використовується безпосередньо для панелей інструментів і як базовий клас для *MenuStrip* і *StatusStrip*. Стандартна функціональність *ToolStrip* - можливість його перетягування в будь-яку область екрану (тоді він відображається як маленьке вікно). Елемент керування *ToolStrip*, що застосовується як панель інструментів, використовує набір елементів керування, що походять від класу *ToolStripItem*.

Ключові властивості компонента *ToolStrip* пов'язані з його позиціонуванням на формі:

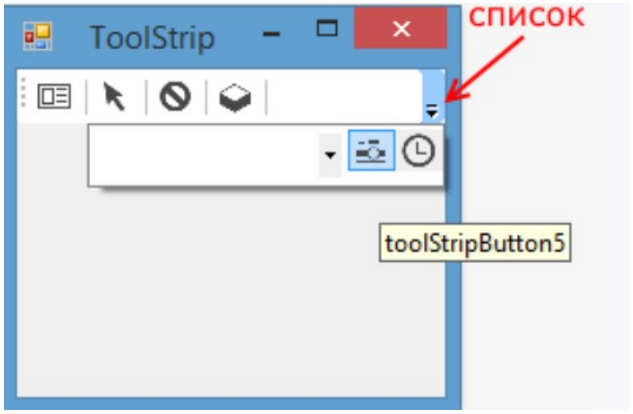
- *Dock*: прикріплює панель інструментів до однієї із сторін форми
- *LayoutStyle*: задає орієнтацію панелі на формі (горизонтальна, вертикальна, таблична)
- *ShowItemToolTips*: вказує, чи відображатимуться підказки для окремих елементів панелі інструментів.
- *Stretch*: дозволяє розтягнути панель по всій довжині контейнера.

В залежності від значення властивості *LayoutStyle* панель інструментів може розташовуватися по горизонталі, або в табличному вигляді:

- *HorizontalStackWithOverflow*: розташування по горизонталі з переповненням - якщо довжина панелі перевищує довжину контейнера, нові елементи, що виходять за межі контейнера, не відображаються, тобто панель переповнюється елементами
- *StackWithOverflow*: елементи розташовуються автоматично з переповненням
- *VerticalStackWithOverflow*: елементи розміщуються вертикально з переповненням
- *Flow*: елементи розташовуються автоматично, але без переповнення - якщо довжина панелі менша за довжину контейнера, то виходять за межі елементи переносяться, а панель інструментів розтягується, щоб вмістити всі елементи
- *Table*: елементи позиціонуються як таблиці:

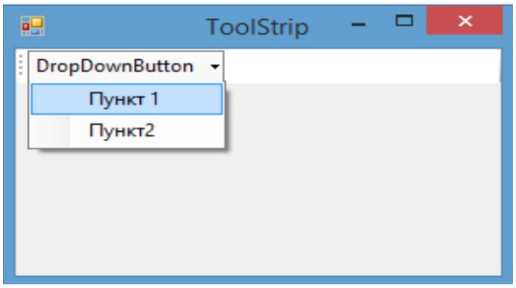


Якщо *LayoutStyle* має значення *HorizontalStackWithOverflow* або *VerticalStack-WithOverflow*, то за допомогою властивості *CanOverflow* ми можемо встановити поведінку при переповненні. Так, якщо ця властивість дорівнює *true* (значення за замовчуванням), то для елементів, що не потрапляють у межі *ToolStrip*, створюється список, що випадає:



При значенні *false* подібний список, що випадає, не створюється. Панель *ToolStrip* може містити такі об'єкти:

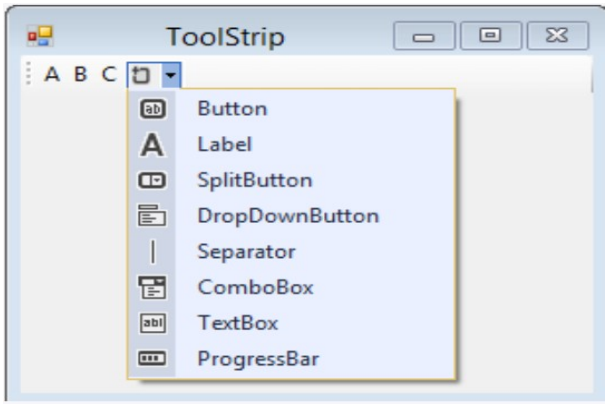
- ToolStripLabel*: текстова мітка на панелі інструментів, представляє функціональність елементів Label та LinkLabel*
- ToolStripButton*: аналогічний елементу Button. Також має подію Click, за допомогою якої можна обробити натискання користувачем кнопки*
- ToolStripSeparator*: візуальний роздільник між іншими елементами на панелі інструментів*
- ToolStripToolStripComboBox*: подібний до стандартного елементу ComboBox*
- ToolStripTextBox*: аналогічний текстовому полю TextBox*
- ToolStripProgressBar*: індикатор прогресу, як і елемент ProgressBar*
- ToolStripDropDownButton*: представляє кнопку, при натисканні по якій відкривається меню, що випадає*:



До кожного елемента меню, що випадає, додатково можна прикріпити обробник натискання та обробити натискання по цих пунктах меню.

- *ToolStripSplitButton*: об'єднує функціональність *ToolStripDropDownButton* та *ToolStripButton*

Додати нові елементи можна в режимі дизайнера:



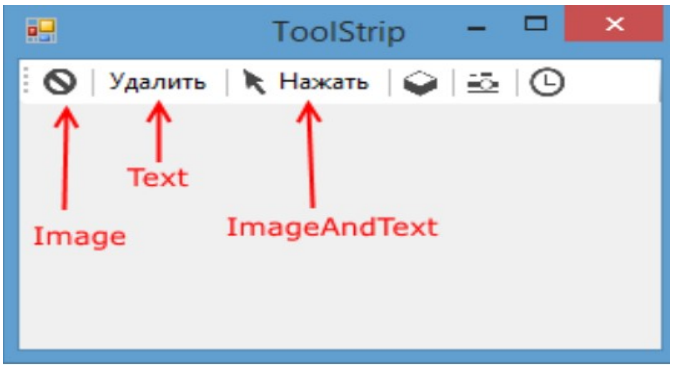
Також можна додавати нові елементи програмно у коді. Їхнє розташування на панелі інструментів буде відповідати порядку додавання. Всі елементи зберігаються в *ToolStrip* як *Items*. Ми можемо додати до нього будь-який об'єкт класу *ToolStripItem* (тобто будь-який з перерахованих вище класів, оскільки вони успадковуються від *ToolStripItem*):

Крім того, тут задається обробник, що дозволяє обробляти натискання кнопки на панелі інструментів.

Елементи *ToolStripButton*, *ToolStripDropDownButton* і *ToolStripSplitButton* можуть відображати як текст, так і зображення або те й інше. Для керування розміщенням зображень у цих елементах є такі властивості:

- DisplayStyle: *визначає, чи відображатиметься на елементі текст, зображення, або те й інше.*
- Image*: вказує на зображення.*
- ImageAlign*: встановлює вирівнювання зображення щодо елемента*
- ImageScaling*: вказує, чи буде зображення розтягуватися, щоб заповнити весь простір елемента*
- ImageTransparentColor*: вказує, чи колір зображення буде прозорим.*

Щоб вказати розмістити зображення на кнопці, у властивості *DisplayStyle* необхідно встановити значення *Image*. Якщо ми хочемо, щоб кнопка відображала тільки текст, треба вказати значення *Text*, або можна комбінувати два значення за допомогою іншого значення *ImageAndText*:



Усі ці значення зберігаються в переліку *ToolStripItem - DisplayStyle.* Також можна встановити властивості в коді *C#*:

```
ToolStripButton clearBtn = new ToolStripButton();
clearBtn.Text = "Поиск";
clearBtn.DisplayStyle = ToolStripItemDisplayStyle.ImageAndText;
clearBtn.Image = Image.FromFile(@"D:\Icons\0023\search32.png");
// добавляем на панель инструментов
toolStrip1.Items.Add(clearBtn)
```

Властивість *DisplayStyle* керує тим, чи відображається текст, зображення, те й інше або ні те, ні інше на поверхні елемента керування. Коли *AutoSize* встановлено в true, *ToolStripItem* буде змінювати свій розмір, тому знадобиться мінімальний простір.

Форматування тексту в *ToolStripItem* керується властивостями *Font*, *TextAlign* та *TextDirection*. Властивість *TextAlign* встановлює вирівнювання тексту щодо елемента керування. Це може бути будь-яке значення з переліку *ControlAlignment*. За замовчуванням приймається значення *MiddleRight*. Властивість *TextDirection* визначає орієнтацію тексту. Значення можуть бути будь-якими з переліку *ToolStripTextDirection*. приклад: *Vertical270*, *Vertical90*. *Vertical270* повертає текст на 270 градусів, а *Vertical90* – на 90 градусів.

Структури меню та панелей інструментів можуть зрости до такого розміру, що ними стає важко керувати. Клас *ToolStripManager* надає можливість створення маленьких, більш керованих фрагментів структур меню або панелі інструментів для того, щоб потім за необхідності їх комбінувати. Прикладом цього може бути форма, що містить кілька елементів керування. Кожен повинен відображати контекстне меню. Декілька пунктів меню повинні бути доступні всім елементам керування, але кожен з них також містить кілька унікальних пунктів. Загальні пункти меню можна визначити в одному *ContextMenuStrip*. Кожен із унікальних пунктів меню може бути визначений заздалегідь або створений під час виконання. Для кожного елемента, якому потрібне контекстне меню, загальне меню клонується і до нього додаються унікальні пункти за допомогою методу *ToolStripManager.Merge*. Результуюче меню призначається властивості *ContextMenuStrip* елемента керування.

Елемент керування *ToolStripContainer* використовується для стикування елементів керування, заснованих на *ToolStrip*. Додавання *ToolStripContainer* і встановлення властивості *Docked* значення *Fill* додає *ToolStripPanel* до кожної сторони форми, а *ToolStripContainerPanel* - у середину форми. Будь-який *ToolStrip* (*ToolStrip*, *MenuStrip* або *StatusStrip*) може бути доданий до будь-якої з панелей *ToolStripPanel*. Користувач може перемістити *ToolStrip* за допомогою миші на будь-яку сторону або нижню частину форми. Встановивши значення *false* властивості *Visible* для будь-якої з панелей *ToolStripPanel*, можна заборонити розміщення панелі *ToolStrip*. Панель *ToolStripContainerPanel* у центрі форми може бути використана для розміщення інших елементів керування, які їй знадобляться.

У наступному прикладі коду показано додавання *ToolStripContainer* і *ToolStrip* до форм *Windows Forms*, додавання елементів до об'єкта *ToolStrip* і додавання *ToolStrip* до властивості *ToolStripPanel* контейнера *ToolStripContainer*.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

public class Form1 : Form
{
    private ToolStripContainer toolStripContainer1;
    private ToolStrip toolStrip1;
    public Form1()
    {
        InitializeComponent();
    }
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }
    private void InitializeComponent()
    {
        toolStripContainer1 = new System.Windows.Forms.ToolStripContainer();
        toolStrip1 = new System.Windows.Forms.ToolStrip();
        // Add items to the ToolStrip.
        toolStrip1.Items.Add("One");
        toolStrip1.Items.Add("Two");
        toolStrip1.Items.Add("Three");
        // Add the ToolStrip to the top panel of the ToolStripContainer.
        toolStripContainer1.TopToolStripPanel.Controls.Add(toolStrip1);
        // Add the ToolStripContainer to the form.
        Controls.Add(toolStripContainer1);
    }
}
```

4.13. Панель стану (елементи керування StatusBar, StatusStrip)

Один з найважливіших аспектів будь-якої програми - інформування користувача про те, що відбувається в програмі в даний момент часу. Наприклад, у текстових редакторах потрібно знати номер поточної сторінки та позицію курсору на сторінці. Подібний тип інформації називається *станом програми*, і зазвичай відображається на екрані постійно в спеціальному рядку (ні про які вікна повідомлень мови, природно, бути не може). Крім того, користувач, напевно, захоче дізнатися, як довго

виконуватиметься конкретна операція, і яка її частина виконана на даний момент. Цей тип інформації називається *ходом виконання програми*.

Для створення панелі стану використовуються елементи керування *StatusBar* та *StatusStrip*, який є більш вдосконаленим і має місце в останніх версіях *C#*.

Однак багато програм використовують *StatusBar* і багато програмістів влаштовують його можливості. Тому доцільно розглянути *StatusBar* та *StatusStrip*.

StatusBar

Для створення рядка стану за допомогою *StatusBar* потрібно вибрати елемент керування *StatusBar* і помістити його у форму. Незалежно від розмірів, які ви спробуєте надати рядку стану, вона розтягнеться на всю ширину форми і міститиме одну панель. Крім того, вона автоматично переміститься у нижню частину форми. Це стандартне розташування рядка стану програми.

Елемент керування *StatusBar* дозволяє одночасно відображати цілий набір різних повідомлень. Кожен розділ у рядку стану називається панеллю. Кількість панелей, що відображаються, визначається, у властивості *Style*, яка може приймати одне з наведених нижче значень:

- *sbrNormal*. Дозволяє відобразити кілька панелей у рядку стану.
- *sbrSimple*. У рядку стану відображається лише одна панель.

Якщо встановити значення властивості *Style* яке буде дорівнювати *sbrSimple*, елемент керування *StatusBar* перетворюється на звичайний напис (елемент керування типу *Label*). Текст, який виводиться на єдину панель, визначається властивістю *SimpleText*. При виборі варіанта рядка стану з кількома панелями стають доступними багато інших можливостей. Зручніше працювати з кількома панелями за допомогою діалогового вікна *Property Pages* рядка стану, показаного на рис. 4.54.

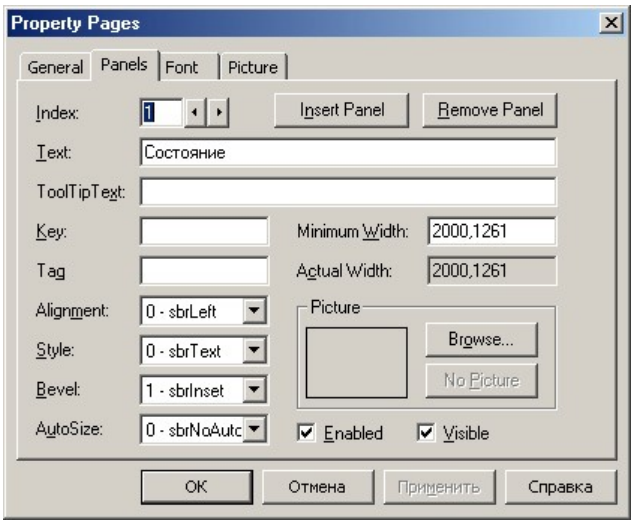


Рис. 4.54. До встановлення властивості *StatusBar*

Для кожної окремої панелі в діалоговому вікні *Property Pages* необхідно визначити вісім основних властивостей, що керують її зовнішнім виглядом та поведінкою.

- *Text*. Визначає текст, який відображатиметься в текстовій панелі. (Зазвичай значення цієї властивості встановлюється під час виконання програми.)
- *ToolTipText*. Текст підказки, яка з'являється, коли користувач затримує курсор миші над панеллю.
- *Alignment*. Керує вирівнюванням тексту на панелі - ліворуч, праворуч або в центрі.
- *Style*. Визначає тип створюваної панелі.
- *Bevel*. Визначає тип затінення для імітації об'ємності панелі.
- *AutoSize*. Визначає принцип керування розміром панелі з програми.
- *MinWidth*. Встановлює мінімальний розмір панелі.
- *Picture*. Зображення, яке можна розмістити на панелі рядка стану.

Хоча більшість властивостей говорять самі за себе, доцільно приділити увагу властивостям *Style* та *AutoSize*.

Залежно від значення властивості *Style*, у рядок стану можна помістити сім різних типів панелей. Одне з них - *sbrText* - призначене для виведення текстових повідомлень, а більшість інших стилів призначені для реалізації стандартних елементів рядка стану, таких як індикатори стану службових клавіш (<*Caps Lock*>, <*Num Lock*> і т.д.), системної дати та часу. Ці зумовлені елементи працюють самі собою. Під час розробки програми потрібно задати лише деякі параметри. Список стилів панелей рядка стану наведено у табл. 4.4.

Таблиця 4.4.

Список стилів панелей рядка стану

Значення	**Опис**
sbrText	Дозволяє відобразити текст або растрове зображення, які зазначені відповідно у властивості *Text* або *picture* панелі рядка стану
sbrCaps	Використовується для індикації стану клавіші <*Caps Lock*>. Коли клавіша активована (тобто включений верхній регістр літер) на панелі рядка стану з'являється текст *CAPS*, виділений темним шрифтом, інакше текст буде набраний затіненим шрифтом
sbrNum	Відображає стан <*Num Lock*>. Подібний до попереднього, тільки на панелі відображається текст *NUM*
sbrIns	Відображає стан <*Insert*>. Подібний до попереднього, тільки на панелі відображається текст *INS*
sbrScrl	Відображає стан <*Scroll Lock*>. Подібний до попереднього, тільки на панелі відображається текст *SCRL*
sbrTime	Призначений для відображення поточного часу
sbrDate	Призначений для відображення поточної дати

Властивість *AutoSize* кожної панелі має три можливі значення, які керують розміром панелі. За замовчуванням задається значення *NoAutoSize*, яке визначає розмір панелі, що дорівнює значенню властивості *MinWidth*; цей розмір не змінюється при додаванні або видаленні інших панелей. При установці значення *sbrContents* розмір панелі залежатиме від її вмісту. Причому під вмістом мається на увазі текст, введений користувачем, інформація про стан клавіш-перемикачів, дата або час. І останнє значення, *sbrSpring*, дозволяє панелям автоматично розширюватися так, щоб заповнити весь простір рядка стану; у цьому випадку у рядку не буде порожніх місць.

Відзначимо ще одну властивість – *Bevel*. Ця властивість задає "об'ємний" зовнішній вигляд панелі. Він має три можливих значення: *sbrNoBevel*, якому відповідає плоска панель; *sbrInsert* (задане за замовчуванням), якому відповідає панель, вдавлена в рядок стану; *sbrRaised*. якому відповідає опукла панель.

Вище було розказано про властивості окремої панелі рядка стану. А тепер поговоримо про додавання та видалення панелей. У діалоговому вікні *Property Pages* для цього призначені кнопки *Insert Panel* і *Remove Panel*. Крім того, додавати та видаляти панелі рядка стану можна з коду програми. Для цього призначено три методи колекції *Panels*, перелічені нижче:

- *Add*. Створює нову панель у рядку стану.
- *Remove*. Видаляє певну панель із рядка стану.
- *Clear*. Видаляє всі панелі з рядка стану.

Ми вже згадували, що панелі, призначені для індикації стану клавіш-перемикачів, дати і часу працюють в автономному режимі. Ще однією перевагою рядка стану є можливість змінювати повідомлення у текстових панелях під час виконання програми. У цих панелях користувачеві зазвичай повідомляється, яку операцію виконує програма зараз. Щоб оновити стан цього елемента в програмі, достатньо присвоїти властивості *Text* об'єкта *Panel* текстовий рядок:

StatusBarl. Panels (1) .Text = "Виведення результатів"

Зрозуміло, так можна змінити й інші властивості панелей. Наприклад, для визначення вмісту рядка стану в коді програми використовуються методи колекції *Panels* та властивості об'єктів *Panel*. У властивості *Count* колекції *Panels* зберігається число панелей рядка стану в даний час.

Важливим елементом відображення процесу виконання операцій є *ProgressBar*. Найважливішими властивостями є *Min* (нижня межа), *Max* (верхня межа) та *Value* (поточне значення).

Визначення значення властивостей *Value* надається розробнику, оскільки сам елемент керування немає можливості відстеження протікання процесу.

Приклад 35.

На лістингу 4.36. наведено програму створення рядка стану (*StatusBar*), що складається з трьох панелей. Після створення форми у першій панелі встановлюється текст. Потім таймер запускається та викликає його подію *Tick*, в обробнику якої встановлюється текст поточної дати та часу відповідно у другій та третій панелях.

На рис. 4.55 показано результат формування рядка стану.

Лістинг 4.36. (examp35)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp35
{
    public partial class Form1 : Form
    {
        StatusBarPanel sbpTime,sbpData;
        //Работа со статусной строкой, временем
        public Form1()
        {
            InitializeComponent();
            Text = "Two Status flar Panels";
            BackColor = SystemColors.Window;
            ForeColor = SystemColors.WindowText;
            StatusBar sb = new StatusBar();
            sb.Parent = this;
            sb.ShowPanels = true;
            //Создание панелей для строки состояния
            StatusBarPanel sbpMenu = new StatusBarPanel();
            sbpMenu.Text = "Reserved for menu help";
            sbpMenu.BorderStyle  = StatusBarPanelBorderStyle.None;
            sbpMenu.AutoSize = StatusBarPanelAutoSize.Contents;
            sbpData = new StatusBarPanel();
            sbpData.AutoSize = StatusBarPanelAutoSize.Contents;
            sbpData.ToolTipText= "The current date";
            sbpTime = new StatusBarPanel();
            sbpTime.AutoSize = StatusBarPanelAutoSize.Contents;
            sbpTime.ToolTipText= "The current time";
            // Сопоставление панелей в строке состояния
             sb. Panels.AddRange(new
StatusBarPanel[]{sbpMenu,sbpData,sbpTime});
             timer1.Interval = 1000;
             timer1.Start();
        }
        private void timer1_Tick(object sender, EventArgs e)
        {

            DateTime dt = DateTime.Now;
            sbpData.Text = dt.ToShortDateString();
            sbpTime.Text = dt.ToShortTimeString();
        }
    }
 }
```

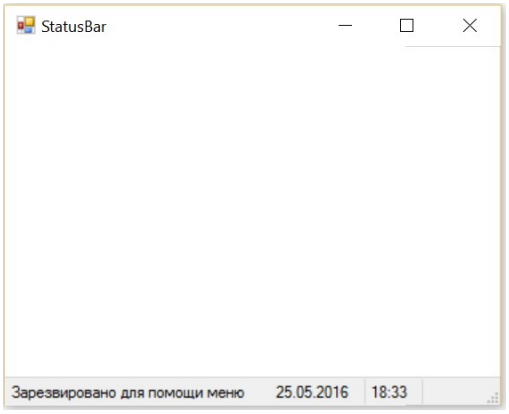


Рис. 4.55. До використання StatusBar.

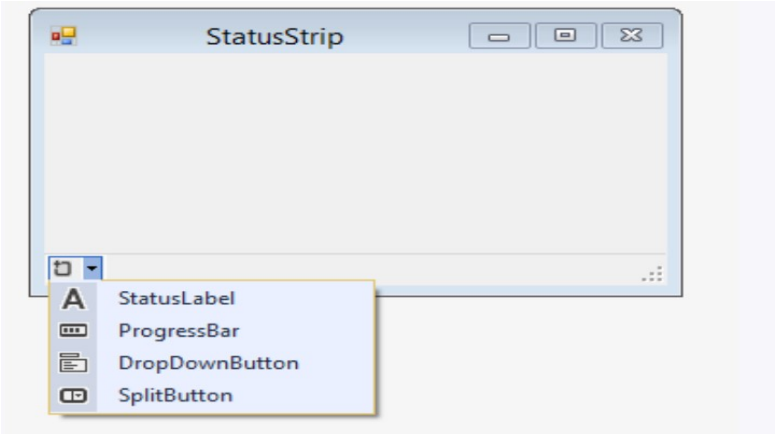
StatusStrip представляє рядок стану, багато в чому аналогічний панелі інструментів *ToolStrip*. Рядок стану призначений для відображення поточної інформації про стан роботи програми.

При додаванні на форму *StatusStrip* автоматично розміщується у нижній частині вікна програми (як і в більшості програм). Однак за необхідності ми зможемо його по-іншому позиціонувати, керуючи властивістю *Dock*, яка може приймати наступні значення:

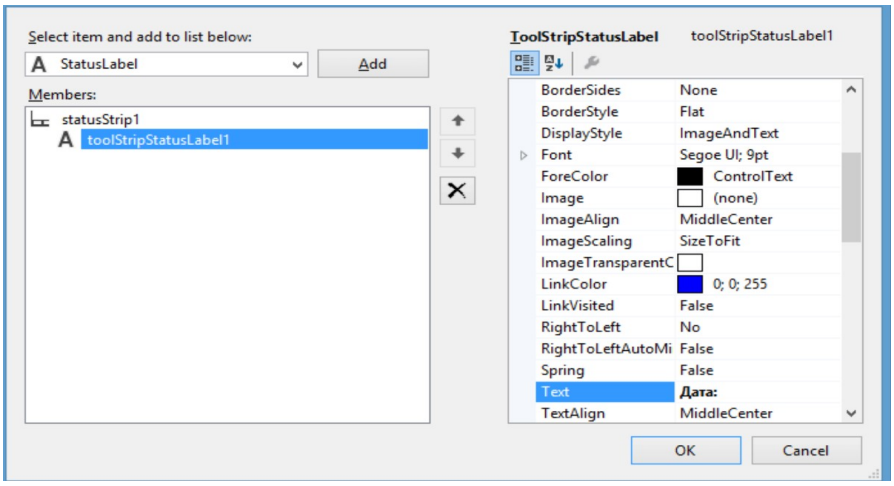
- *Bottom*: розміщення внизу (за замовчуванням)
- *Top*: прикріплює статусний рядок до верхньої частини форми
- *Fill*: розтягує всю форму
- *Left*: розміщення у лівій частині форми
- *Right*: розміщення у правій частині форми
- *None*: довільне становище

StatusStrip може містити різні елементи. У режимі дизайнера ми можемо додати такі типи елементів:

- *StatusLabel*: мітка для виведення текстової інформації. Представляє об’єкт *ToolStripLabel*
- *ProgressBar*: індикатор прогресу. Представляє об’єкт *ToolStripProgressBar*
- *DropDownButton*: кнопка зі списком по кліку. Представляє об’єкт *ToolStripDropDownButton*
- *SplitButton*: ще одна кнопка багато в чому аналогічна *DropDownButton*. Представляє об’єкт *ToolStripSplitButton*



Або можна звернутися на панелі властивостей до властивості *Items* компонента *StatusStrip* та у вікні додати і налаштувати всі елементи:



Також ми можемо додати елементи програмно.

Приклад 36.

Створимо невелику програму, наведену у лістингу 4.37. Умова завдання така сама, як і в попередньому прикладі при використанні *StatusBar*. Відмінність в тому, що в рядку стану додано мітку *ProgressBar*. Після створення форми таймер запускається і викликає його подію *Tick*, в обробнику якої встановлюємо текст міток. Крім того виконується заповнення *ProgressBar*. На рис. 4.56. наведено результат виконання програми.

Лістинг 4.37. (examp36)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace examp36 {
    public partial class Form1 : Form {
        Timer timer;
        int val;
        public Form1() {
            InitializeComponent();
            val = 0;
            infoLab.Text = "Текущие дата и время:";
            ProgressBar.Minimum = 0;
            ProgressBar.Maximum = 100;
            ProgressBar.Value = 0;
            timer = new Timer() { Interval = 1000 };
            timer.Tick += timer_Tick;
            timer.Start();
        }
        void timer_Tick(object sender, EventArgs e) {
            dateLab.Text = DateTime.Now.ToLongDateString();
            timeLab.Text = DateTime.Now.ToLongTimeString();
            if (val < 100) val += 5;
```

```
                ProgressBar.Value = val;
            }
        }
    }
```

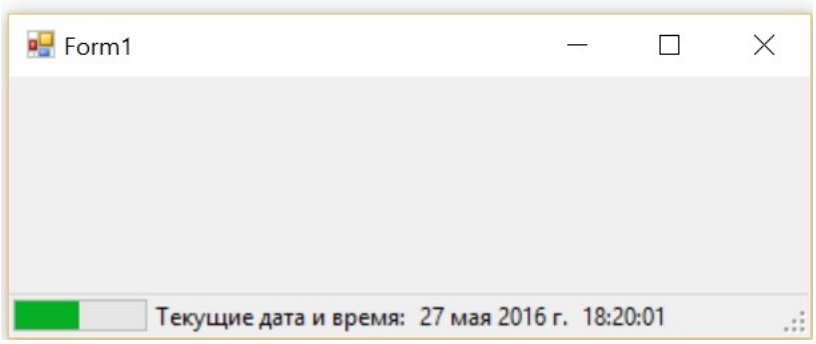


Рис. 4.56. До використання StatusStrip

4.14. Елементи керування Splitter та SplitContainer

Елемент керування Splitter

Елемент керування *Splitter* дозволяє змінювати розміри елементів керування, прикріплених до країв елемент керування *Splitter*, під час виконання програми. Коли користувач поміщає вказівник миші на елемент керування *Splitter*, курсор змінює свій вигляд, що є ознакою того, що розміри елементів керування, прикріплених до елемента керування *Splitter*, можуть бути змінені. Елемент *Splitter* дозволяє змінювати розмір елемента керування, приєднаного безпосередньо перед ним. Таким чином, щоб дозволити користувачу змінювати розмір прикріпленого елемента керування, слід прикріпити даний елемент керування до краю контейнера, а потім до тієї сторони контейнера прикріпити роздільник. Наприклад, щоб створити вікно подібне до провідника, слід додати до форми елемент керування *TreeView* і встановити його властивість *Dock* рівною *DockStyle.Left*. Потім потрібно додати до форми елемент керування *Splitter* і встановити його властивість *Dock* рівною *DockStyle.Left*. Для завершення компонування форми слід додати елемент керування *ListView* і встановити його властивість *Dock* рівною *DockStyle.Fill* для того, щоб *ListView* займав весь простір форми, що залишився. Тепер під час виконання користувач може змінювати ширину елемента керування *TreeView* (як і *ListView*), переміщуючи елемент керування *Splitter*.

Щоб запобігти можливості зменшення елементом керування *Splitter* розміру прикріплених елементів керування настільки, що це стане неприйнятним для користувача, слід задіяти властивості *MinExtra* і *MinSize*. Властивості *MinExtra* та *MinSize* визначають мінімальний розмір, до якого можна зменшити елементи керування, прикріплені ліворуч та праворуч (або зверху та знизу, якщо використовується горизонтально орієнтований елемент керування *Splitter*). Якщо елементи керування форми, до яких прикріплено елемент керування Splitter, відображаються зі специфічною рамкою, для встановлення відповідності стилів рамок прикріплених елементів можна скористатися властивістю *BorderStyle*.

Можлива ситуація, коли для елементів керування, до яких приєднано елемент керування *Splitter*, потрібно встановити найбільший допустимий розмір. Події *SplitterMoved* та *SplitterMoving* дають можливість визначити момент, коли користувач змінює розмір прикріпленого елемента. Для визначення розміру елемента керування, до якого прикріплений елемент керування *Splitter*, у обробниках подій *SplitterMoved* та *SplitterMoving* можна скористатися властивістю *SplitPosition*. При цьому можна змінювати значення властивості *SplitPosition* для того, щоб не допустити перевищення заданої найбільшої ширини (або висоти, у разі горизонтально орієнтованого елемента керування *Splitter*).

Зміна розміру елемента керування за допомогою елемента керування *Splitter* можлива лише за допомогою миші. Доступ до елемента керування *Splitter* неможливий за допомогою клавіатури.

Приклад 37.

Розглянемо приклад поділу форми на три області за допомогою двох роздільників, як це показано на рис. 4.57. При цьому кожна з областей є панелью. Необхідно надати код виведення еліпсів, вписаних у панелі. Доцільно поділ форми на панелі виконувати у режимі дизайнера, тобто. під час проектування. Спочатку на форму поміщаємо першу панель і встановимо її властивість *Dock*, що дорівнює *DockStyle.Top*. Потім поміщається *Splitter* з такою самою властивістю *Dock*. Потім встановлюємо другу панель з властивістю *Dock*, що дорівнює *DockStyle.Left*. Потім розміщується *Splitter* з такою ж властивістю. Для третьої панелі *Dock* присвоюємо *Dock* значення *DockStyle.Fill* завдання, для того, щоб ця панель займала весь простір форми, що залишився. Колір панелей та роздільників, ширина смуг роздільників, а також відгуки виконуються у дизайнері. Тому доцільно навести код функції *InitializeComponent()*. Цей код наведено у лістингу 4.38.

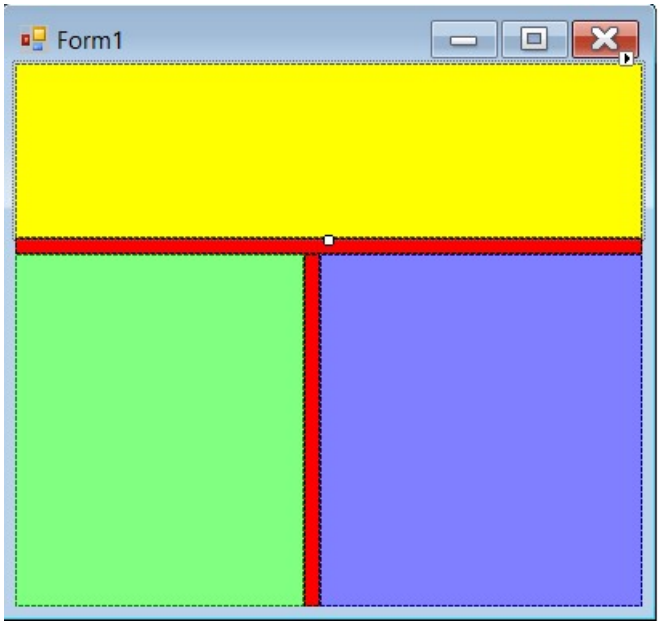


Рис. 4.57. Розділ форми за допомогою роздільників

Лістинг 4.38.

```
private void InitializeComponent()
        {
            this.panel1 = new System.Windows.Forms.Panel();
            this.splitter1 = new System.Windows.Forms.Splitter();
            this.panel2 = new System.Windows.Forms.Panel();
            this.splitter2 = new System.Windows.Forms.Splitter();
            this.panel3 = new System.Windows.Forms.Panel();
            this.SuspendLayout();
            //
            // panel1
            //
            this.panel1.BackColor = System.Drawing.Color.Yellow;
            this.panel1.Dock = System.Windows.Forms.DockStyle.Top;
            this.panel1.Location = new System.Drawing.Point(0, 0);
            this.panel1.Name = "panel1";
            this.panel1.Size = new System.Drawing.Size(391, 109);
            this.panel1.TabIndex = 0;
            this.panel1.Paint += new System.Windows.Forms. PaintEventHandler
            (this.panel_Paint);
            this.panel1.Resize += new System.EventHandler(this.panel_Resize);
            //
            // splitter1
            //
            this.splitter1.BackColor = System.Drawing.Color.Red;
            this.splitter1.Dock = System.Windows.Forms.DockStyle.Top;
            this.splitter1.Location = new System.Drawing.Point(0, 109);
            this.splitter1.Name = "splitter1";
            this.splitter1.Size = new System.Drawing.Size(391, 10);
            this.splitter1.TabIndex = 1;
            this.splitter1.TabStop = false;
            //
            // panel2
            //
            this.panel2.BackColor =
System.Drawing.Color.FromArgb(128,128,255)));
            this.panel2.Dock = System.Windows.Forms.DockStyle.Left;
            this.panel2.Location = new System.Drawing.Point(0, 119);
            this.panel2.Name = "panel2";
            this.panel2.Size = new System.Drawing.Size(180, 220);
            this.panel2.TabIndex = 2;
            this.panel2.Paint += new System.Windows.Forms.PaintEventHandler
            (this.panel_Paint);
            this.panel2.Resize += new System.EventHandler(this.panel_Resize);
            //
            // splitter2
            //
            this.splitter2.BackColor = System.Drawing.Color.Red;
            this.splitter2.Location = new System.Drawing.Point(180, 119);
            this.splitter2.Name = "splitter2";
            this.splitter2.Size = new System.Drawing.Size(10, 220);
            this.splitter2.TabIndex = 3;
            this.splitter2.TabStop = false;
            //
            // panel3
            //
            this.panel3.BackColor = System.Drawing.Color.FromArgb(128,128,255);
            this.panel3.Dock = System.Windows.Forms.DockStyle.Fill;
            this.panel3.Location = new System.Drawing.Point(190, 119);
            this.panel3.Name = "panel3";
            this.panel3.Size = new System.Drawing.Size(201, 220);
            this.panel3.TabIndex = 4;
```

```
            this.panel3.Paint += new System.Windows.Forms.PaintEventHandler
            (this.panel_Paint);
            this.panel3.Resize += new System.EventHandler(this.panel_Resize);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(391, 339);
            this.Controls.Add(this.panel3);
            this.Controls.Add(this.splitter2);
            this.Controls.Add(this.panel2);
            this.Controls.Add(this.splitter1);
            this.Controls.Add(this.panel1);
            this.Margin = new System.Windows.Forms.Padding(4);
            this.Name = "Form1";
            this.Text = "Form1";
            this.ResumeLayout(false);
        }
        #endregion
        private System.Windows.Forms.Panel panel1;
        private System.Windows.Forms.Splitter splitter1;
        private System.Windows.Forms.Panel panel2;
        private System.Windows.Forms.Splitter splitter2;
        private System.Windows.Forms.Panel panel3;
    }
}
```

Код програми наведено у лістингу 4.39. Обсяг коду незначний, оскільки основна робота виконана у *Designer*. Результат рішення подано на рис. 4.58.

Лістинг 4.39. (examp37)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp37
{
    //Работа с элементом Splitter
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            Text = "One Panel with Splitter";

        }

        private void panel_Paint(object sender, PaintEventArgs e)
        {
            Panel panel = (Panel)sender;
            e.Graphics.DrawEllipse(new Pen(Color.Black, 2), 0, 0, panel.Width -
2,
            panel.Height - 2);
        }
```

```
        private void panel_Resize(object sender, EventArgs e)
        {
            ((Panel)sender).Invalidate();
        }
    }
}
```

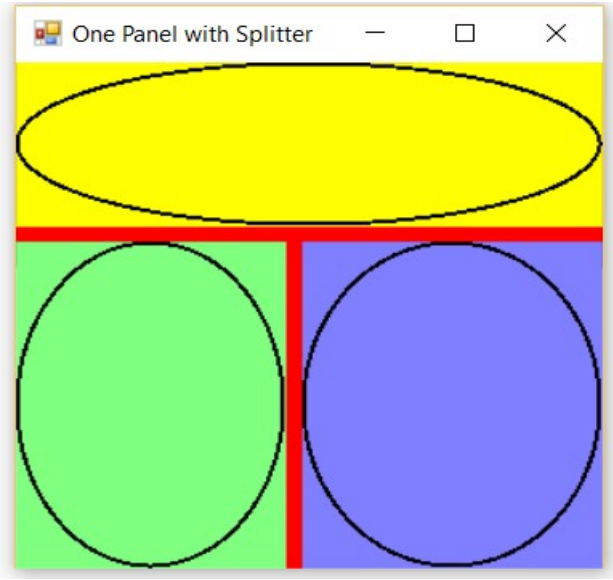


Рис. 4.58. Виведення еліпсів у панелях

Приклад 38.

Розглянуте завдання можна реалізувати програмним кодом, без використання дизайнера. Саме це використовувалося у старих версіях *Visual Studio*. Це можна привести цей код, оскільки іноді виникає необхідність змінювати властивості елементів керування під час виконання програми. Код представлений у лістингу 4.40. При цьому розглядається ще один варіант встановлення значень властивостей *Dock* для панелей та роздільників.

Лістинг 4.40. (examp38)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp38
{
    //Работа с элементом Splitter
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            Text = "One Panel with Splitter";

            Panel panel1 = new Panel();
            panel1.Parent = this;
```

```
            panel1.Dock = DockStyle.Fill;
            panel1.BackColor = Color.Blue;
            panel1.Resize += new EventHandler(PanelOnResize);
            panel1.Paint += new PaintEventHandler(PanelOnPaint);

            Splitter split = new Splitter();
            split.Parent = this;
            split.Dock = DockStyle.Left;
            split.BackColor = Color.Red;

            Panel panel2 = new Panel();
            panel2.Parent = this;
            panel2.Dock = DockStyle.Left;
            panel2.BackColor = Color.Yellow;
            panel2.Resize += new EventHandler(PanelOnResize);
            panel2.Paint += new PaintEventHandler(PanelOnPaint);

            Splitter split2 = new Splitter();
            split2.Parent = this;
            split2.Dock = DockStyle.Top;
            split2.BackColor = Color.Red;

            Panel panel3 = new Panel();
            panel3.Parent = this;
            panel3.Dock = DockStyle.Top;
            panel3.BackColor = Color.Tan;
            panel3.Resize += new EventHandler(PanelOnResize);
            panel3.Paint += new PaintEventHandler(PanelOnPaint);

            panel2.Width = (int)(ClientSize.Width*0.6);
            panel3.Height = ClientSize.Height / 3;
        }

        void PanelOnResize(object obj, EventArgs ea)
        {
           ((Panel)obj).Invalidate();
        }

        void PanelOnPaint (object obj, PaintEventArgs pea)
     {
        Panel panel = (Panel)obj;
        Graphics gr = pea.Graphics;
        Pen pen = new  Pen(Color.Black,3);
        gr.DrawEllipse(pen, 0, 0,
        panel.Width - 1, panel.Height - 1);
      }
    }
}
```

Елемент керування SplitContainer

Елемент керування *SplitContainer* складається з двох панелей, розділених рухомою смугою. При наведенні курсора миші на смугу його форма змінюється, показуючи, що смуга переміщається.

На панелі елементів цей елемент керування замінює елемент *Splitter*, який був у попередній версії *Visual Studio*. Елемент керування *SplitContainer* краще, ніж елемент керування *Splitter*. За допомогою елемента керування *SplitContainer* можна створювати складні інтерфейси користувача. Часто вибір на одній панелі визначає об’єкти, які відображаються на іншій. Такий

підхід є дуже ефективним для відображення та перегляду інформації. Дві панелі дозволяють групувати інформацію, а смуга чи роздільник спрощують зміну розміру панелей.

Приклад 39.

У наступному прикладі, як і попередніх з використанням *Splitter*, ми розділимо форму на три області (панелі) і в кожній з них намалюємо еліпс. У цьому прикладі показаний горизонтальний та вертикальний *SplitContainer*. Цей елемент генерує свої панелі, тому немає потреби формування цих об'єктів. Вертикальне чи горизонтальне розташування розглянутого елемента визначається властивістю *Orientation*. У цьому прикладі використано відгук *Paint* для відтворення еліпсів у панелях. Переміщення вертикального роздільника викликає подію *SplitterMoving*, що відображається в цьому прикладі зміною стилю курсору. Код програми наведено у лістингу 4.41. Тут основний код формується дизайнером і наведено у функції *InitializeComponent()*. Результат роботи програми наведено на рис. 4.59.

Лістинг 4.41. (examp39)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp39
{
    public partial class Form1 : Form
    {
        private System.Windows.Forms.SplitContainer splitContainer1;
        private System.Windows.Forms.SplitContainer splitContainer2;
        public Form1() {
           InitializeComponent();
         }
            private void InitializeComponent() {
                this.splitContainer1 = new
System.Windows.Forms.SplitContainer();
                this.splitContainer2 = new
System.Windows.Forms.SplitContainer();
                ((System.ComponentModel.ISupportInitialize)
                (this.splitContainer1)).BeginInit();
                this.splitContainer1.Panel2.SuspendLayout();
                this.splitContainer1.SuspendLayout();
                ((System.ComponentModel.ISupportInitialize)
                (this.splitContainer2)).BeginInit();
                this.splitContainer2.SuspendLayout();
                this.SuspendLayout();
                //
                // splitContainer1
                //
                this.splitContainer1.BackColor = System.Drawing.Color.Red;
                this.splitContainer1.Dock =
System.Windows.Forms.DockStyle.Fill;
```

```
                    this.splitContainer1.ForeColor = System.Drawing.Color.Red;
                    this.splitContainer1.Location = new System.Drawing.Point(0, 0);
                    this.splitContainer1.Name = "splitContainer1";
                    this.splitContainer1.Orientation =
                    System.Windows.Forms.Orientation.Horizontal;
                    //
                    // splitContainer1.Panel1
                    //
                    this.splitContainer1.Panel1.BackColor =
                    System.Drawing.Color.YellowGreen;
                 this.splitContainer1.Panel1.ForeColor =
System.Drawing.Color.Silver;
                    this.splitContainer1.Panel1.Paint +=
                    new System.Windows.Forms.PaintEventHandler
                    (this.splitContainer_Panel_Paint);
                    //
                    // splitContainer1.Panel2
                    //
                    this.splitContainer1.Panel2.Controls.Add(this.splitContainer2);
                 this.splitContainer1.Panel2.ForeColor =
System.Drawing.Color.DimGray;
                    this.splitContainer1.Panel2.Paint +=
                    new System.Windows.Forms.PaintEventHandler
                    (this.splitContainer_Panel_Paint);
                    this.splitContainer1.Size = new System.Drawing.Size(691, 440);
                    this.splitContainer1.SplitterDistance = 189;
                    this.splitContainer1.TabIndex = 0;
                    this.splitContainer1.SplitterMoving +=
                    new System.Windows.Forms.SplitterCancelEventHandler
                    (this.splitContainer_SplitterMoving);
                    //
                    // splitContainer2
                    //
                    this.splitContainer2.Dock =
System.Windows.Forms.DockStyle.Fill;
                    this.splitContainer2.Location = new System.Drawing.Point(0, 0);
                    this.splitContainer2.Name = "splitContainer2";
                    //
                    // splitContainer2.Panel1
                    //
                    this.splitContainer2.Panel1.BackColor =
System.Drawing.Color.Tan;
                    this.splitContainer2.Panel1.Paint +=
                    new System.Windows.Forms.PaintEventHandler
                    (this.splitContainer_Panel_Paint);
                    this.splitContainer2.Panel2.Paint +=
                    new System.Windows.Forms.PaintEventHandler
                    (this.splitContainer_Panel_Paint);
                    //
                    // splitContainer2.Panel2
                    //
                this.splitContainer2.Panel2.BackColor =
System.Drawing.Color.SkyBlue;
                    this.splitContainer2.Size = new System.Drawing.Size(691, 247);
                    this.splitContainer2.SplitterDistance = 324;
                    this.splitContainer2.TabIndex = 0;
                    this.splitContainer2.SplitterMoving +=
                    new System.Windows.Forms.SplitterCancelEventHandler
                    (this.splitContainer_SplitterMoving);
                    //
                    // Form1
                    //
```

```
            this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(691, 440);
            this.Controls.Add(this.splitContainer1);
            this.Name = "Form1";
            this.Text = "Form1";
            this.splitContainer1.Panel2.ResumeLayout(false);
            ((System.ComponentModel.ISupportInitialize)
            (this.splitContainer1)).EndInit();
            this.splitContainer1.ResumeLayout(false);
            ((System.ComponentModel.ISupportInitialize)
            (this.splitContainer2)).EndInit();
            this.splitContainer2.ResumeLayout(false);
            this.ResumeLayout(false);
        }
                    private void splitContainer_SplitterMoving
        (object sender, SplitterCancelEventArgs e)
        {
           Cursor.Current = System.Windows.Forms.Cursors.NoMoveVert;
        }
        private void splitContainer_Panel_Paint
        (object sender, PaintEventArgs e)
        {
            Panel panel = (Panel)sender;
            Graphics gr = e.Graphics;
            Pen pen = new Pen(Color.Black, 3);
            gr.DrawEllipse(pen, 0, 0,
            panel.Width - 1, panel.Height - 1);
        }
    }
}
```

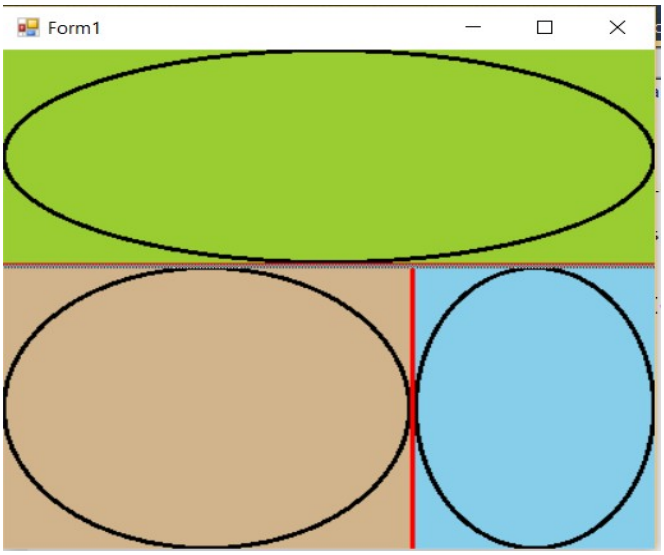


Рис. 4.59. До використання елемента керування SplitContainer

4.15. Список ListView

Елемент керування *ListView* має розширені можливості відображення списків різних об’єктів, що мають лінійну структуру. У нашому випадку цей елемент керування буде використаний для відображення файлів. Розглянемо використання *ListView* на наступному прикладі. Ставиться завдання виведення у вигляді таблиці даних по рядках та стовпцям. Кожен рядок

містить дані про файли. Усього три колонки. Перший стовпець містить ім’я файлу, другий – розмір файлу, а третій – дату зміни файлу. При цьому перед кожним рядком наводиться точковий рисунок. Головною метою розробки є показати роботу *ListView*, тому для спрощення задачі дані файлів прийняті у вигляді масивів *string*. Результат розв’язання задачі подано на рис. 4.60.

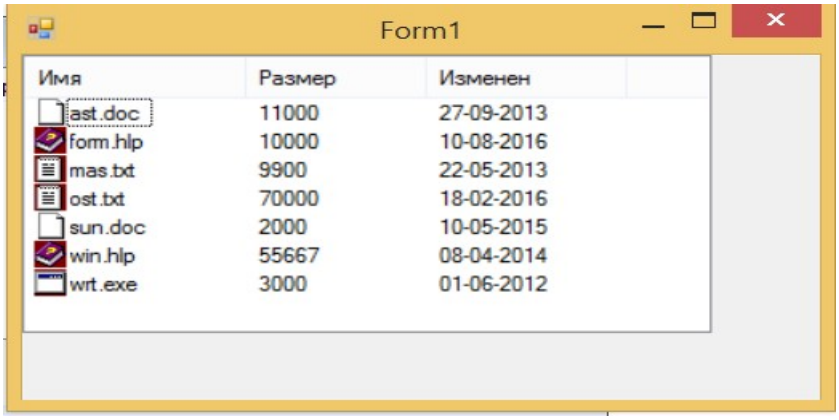


Рис. 4.60. До використання ListView

Послідовність розв’язання цієї задачі полягає в наступному. Перетягуємо піктограму елемента керування *ListView* у вікно програми. При цьому дизайнер форм автоматично створює код, який створює список об’єктів класу *ListView*. Для цього необхідно налаштувати деякі властивості списку. Насамперед, необхідно налаштувати властивість *View*. Ця властивість визначає, у якому вигляді відображатиметься список. Якщо значення цієї властивості дорівнює *Details*, ми побачимо вміст списку як деталізованої таблиці. Саме це значення буде використано у нашому прикладі. Значення *List* визначає режим відображення у вигляді простого списку, а значення *SmallIcon* і *LargeIcon* - у вигляді списку зі значками маленького і великого розміру, відповідно.

Отже, насамперед виділимо список *ListView* у вікні дизайнера форм і встановимо для властивості *View* значення *Details*. За потреби програма зможе динамічно перемикати режими відображення, змінюючи значення *View* під час своєї роботи. Оскільки ми будемо відображати список у вигляді таблиці, нам необхідно створити стовпці таблиці та визначити їх атрибути. Для цього необхідно відредагувати властивість *Columns*. Це робиться за допомогою редактора *ColumnHeader Collection Editor*, який показано на рис. 4.61.

За допомогою кнопки *Add* додаємо в таблицю три стовпці, а потім налаштуємо властивості *Text*, *TextAlign* та *Width* цих стовпців. Властивість *Text* визначає назву стовпця, що відображається у верхній частині списку *ListView*. Перший стовпець (з індексом 0) має називатися Ім’я, другий – Розмір, а третій – Змінено. У стовпці Ім’я ми показуватимемо імена файлів. Що ж до стовпців Розмір і Змінено, то перший з них призначений для відображення розмірів файлів, а другий - для відображення дати останньої зміни файлу.

Рис. 4.61. Редактор стовпців списку

Властивість *TextAlign* визначає вирівнювання заголовка стовпця. За замовчуванням ця властивість має значення *Left*, що задає вирівнювання лівою межею. За потреби Ви можете вирівняти текст з правого боку, вказавши значення *Right*, або вивести заголовки по центру за допомогою *Center*.

За допомогою властивості *Width* можна вказати початкову ширину стовпчика в пікселях. Після відображення списку, користувач зможе змінювати ширину стовпців за допомогою миші. Властивість *Sorting*, що задає сортування списку, може мати значення *None*, *Ascending* та *Descending*.

У першому випадку список відображає елементи в порядку, в якому вони були додані. Значення *Ascending* визначає сортування в порядку зростання, а значення *Descending* - в порядку зменшення.

Як ми вже говорили, елемент керування *ListView* може відображати вміст свого вікна в чотирьох різних режимах, які задаються за допомогою властивості *View*. Залежно від вибраного режиму, елементи списку можуть забезпечуватись піктограмами маленького або великого розміру.

Як видно на рис. 4.60 ми використовуємо 4 значки для відображення файлів різних типів (файли програм, файлів довідкової системи, текстових файлів, файлів документів тощо).

Для роботи з цими значками нам знадобиться список зображень *ImageList*. Перетягніть його значок з панелі *Toolbar* у вікно дизайнера форм. У попередніх прикладах *ImageList,* картинки завантажувалися з ресурсів. У цьому випадку це можна зробити за допомогою властивості *Images*. При натисканні миші з’являється діалогове вікно, наведене на рис. 4.62. При натисканні кнопки *Add* виконується завантаження файлів зображень. Кожна картинка має свій індекс для підключення до списку програмного режиму. У нашому випадку підключено 6 картинок з індексами 0-5.

При завантаженні файлів зазвичай використовуються спеціальні папки. У нашому випадку для операційної системи *Windows 8.1* та використання *Visual Studio 10* це папки:

Program Files (x86)\Microsoft Visual Studio\Common\Graphics\Bitmaps\ Outline\NoMask.

З цієї папки копіюються піктограми *CLSDFOLD.BMP, DOC.BMP, EXE.BMP, HLP.BMP, TXT.BMP* та *WINDOC.BMP*.

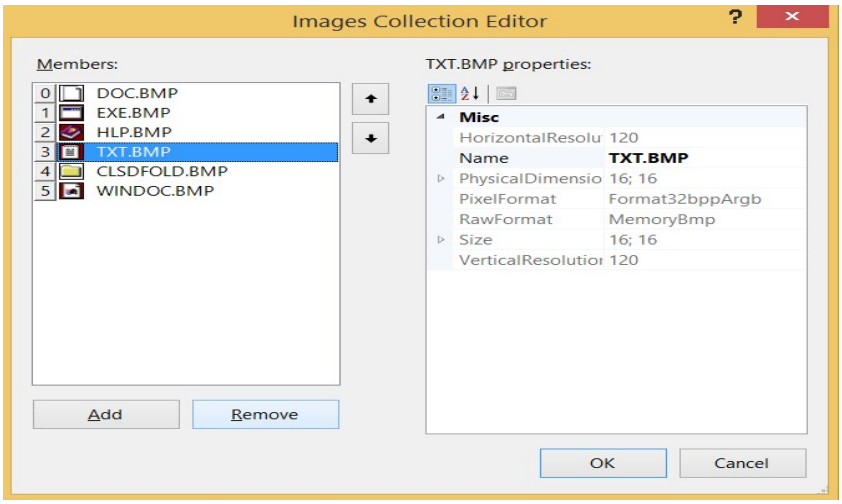


Рис. 4.62. Список зображень для списку каталогів та файлів

У нашому додатку список *ListView* відображається лише в одному режимі, а саме як детальна таблиця. Така таблиця забезпечується значками лише невеликого розміру. Якщо Ваша програма буде використовувати режим *LargeIcon*, необхідно підготувати додатковий список *ImageList*, додавши до нього файли значків великого розміру.

Підготувавши список зображень *ImageList* (ідентифікатор *imageList1*), підключається його до елемента керування *ListView*. Для цього надається властивостям *SmallImageList*, *LargeImageList* значення *imageList1*.

У тому випадку, якщо список *ListView* буде показувати значки і маленьких, і великих розмірів, привласніть ідентифікатор списку зображень маленького розміру *SmallImageList*, а властивості *LargeImageList* - ідентифікатор списку зображень великого розміру.

У загальному випадку список *ListView* містить елементи, кожен із яких має додаткові елементи нижчого рівня, які ми називатимемо атрибутами. Сам елемент представляється текстовим рядком (і, можливо, значком) у першому стовпці списку, що відображається у вигляді деталізованої таблиці. Значення інших атрибутів відображаються в інших стовпцях таблиці (можна також створювати атрибути елементів списку, що не відображаються, що містять довільні дані).

Алгоритм заповнення списку досить простий. Насамперед, необхідно очистити список від попереднього вмісту, тому що наповнення того самого списку може відбуватися неодноразово. Ця операція виконується за допомогою методу *Clear* властивості *Items* списку:

```
listView1.Items.Clear();
```

Далі, щоб додати елемент до списку, потрібно створити новий елемент як об’єкт класу *ListViewItem*:

```
ListViewItem a1 = new ListViewItem("sun.doc");
```

Як параметр конструктору передається текстовий рядок імені файлу. Він відображається у першому стовпчику деталізованої таблиці.

Далі слід додати атрибути елемента. Ця операція виконується за допомогою методу *SubItems.Add*:

```
a1.SubItems.Add("2000");
a1.SubItems.Add("10-05-2015");
```

Тут ми додаємо до елемента два атрибути, перший з яких є розміром файлу, а другий - датою його створення.

Якщо до елемента керування *ListView* додано списки зображень значків, то під час створення елемента списку необхідно вказати індекс потрібного значка. Це робиться за допомогою властивості *ImageIndex*:

```
a1.ImageIndex = 0;
```

Після того, як усі атрибути елемента визначено, Ви можете додати елемент до списку за допомогою методу *Add*:

```
listView1.Items.Add(lvi);
```

Приклад 40.

Повний текс програми наведено у лістингу 4.42. При відгуку на натискання по елементу списку передбачено виведення даних вибраного файлу.

Лістинг 4.42. (examp40)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp40
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            listView1.Items.Clear();
            string[] str1 = { "sun.doc", "wrt.exe", "form.hlp", "ost.txt",
            "mas.txt", "ast.doc", "win.hlp" };
            string[] str2 = { "2000", "3000", "10000", "70000", "9900",
            "11000", "55667" };
            string[] str3 = { "10-05-2015", "01-06-2012", "10-08-2016",
            "18-02-2016", "22-05-2013", "27-09-2013", "08-04-2014" };
            int[] ind = { 0, 1, 2, 3, 3, 0, 2 };
            ListViewItem a1;
            for (int i = 0; i < 7; i++)
            {
                a1 = new ListViewItem(str1[i]);
```

```
                    a1.SubItems.Add(str2[i]);
                    a1.SubItems.Add(str3[i]);
                    a1.ImageIndex = ind[i];
                    listView1.Items.Add(a1);
                }
            }

            private void listView1_ItemActivate(object sender, EventArgs e)
            {
                foreach (ListViewItem a1 in listView1.SelectedItems)
                {
                  string str = "Номер строки = " + (a1.Index + 1) + " Имя - " +
                  a1.SubItems[0].Text + " Размер - " + a1.SubItems[1].Text + " Дата
- " +
                  a1.SubItems[2].Text;
                  MessageBox.Show(str);
                }
            }
        }
    }
```

4.16. Елемент керування TreeView

Елемент керування *TreeView* (для простоти ми будемо називати його деревом *TreeView*) є дуже зручним засобом відображення ієрархічно організованої інформації. Після додавання елементів керування *TreeView* у вікно додатків, ви можете за необхідності відрегулювати початкову ширину вікна, яку він матиме відразу після запуску програми. У нашій програмі вікно елемента керування *TreeView* буде використано для відображення списку дисків і каталогів (аналогічно дереву, розташованому в лівій частині головного вікна програми *Windows Explorer*). Клас *TreeView* досить складний. Він містить безліч методів, властивостей та створює різні події. Разом із класом *TreeView* ми будемо використовувати клас *TreeNode*, який містить записи вузлів дерева. Тобто дерево містить вузли, які представляють об’єкти *TreeNode*. Розглянемо роботу *TreeView* на конкретному прикладі.

Приклад 41.

Перетягуємо з панелі *Toolbox* у вікно форми послідовно три значки: дерево *TreeView*, роздільник *Splitter* та *Panel*. При цьому в методі *InitializeComponent()* створено відповідно три об’єкта:

```
private System.Windows.Forms.TreeView treeView1;
private System.Windows.Forms.Splitter splitter1;
private System.Windows.Forms.Panel panel1;
```

Для поділу форми на дві частини дерева (ліворуч) та панелі (праворуч), як це показано на рис. 4.63., визначені в режимі проектування такі властивості *Dock*:

```
this.treeView1.Dock = System.Windows.Forms.DockStyle.Left;
this.splitter1.Dock = System.Windows.Forms.DockStyle.Left;
this.panel1.Dock = System.Windows.Forms.DockStyle.Fill;
```

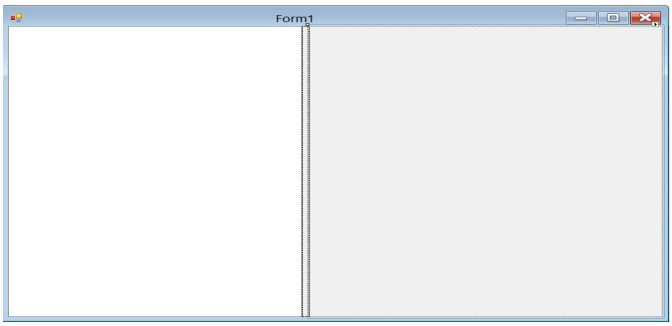


Рис. 4.63. До розділу елементів керування роздільником

Ініціалізація дерева виконується у конструкторі класу *Form1* за допомогою методу *DriveTreeInit*. Крім того, тут встановлюється колір роздільника:

```
public Form1()
{
InitializeComponent();
splitter1.BackColor = Color.Red;
DriveTreeInit()
}
```

Нижче ми опишемо вихідний текст цього методу, а поки що трохи відволічемося - розповімо про класи та методи, що дозволяють отримати інформацію про диски та каталоги.

Щоб заповнити дерево, нам потрібно, перш за все, отримати список логічних дисків у системі. Для цього скористайтесь статичним методом *Directory.GetLogicalDrives*:

```
string[]drivesArray = Directory.GetLogicalDrives();
```

Цей метод немає параметрів. Після виконання він повертає посилання на масив текстових рядків виду «*C:*» з позначками всіх доступних логічних дискових пристроїв.

Щоб звернутися до цього методу, а також до інших методів, які працюють з дисками, каталогами та файлами, підключіть простір імен *System.IO*: *using System.IO*;

Лістинг 4.43. (examp41)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace examp41
{
    public partial class Form1 : Form
    {
```

```
  // Panel panel;
   TreeNode tnSelect;
   TreeNode selectedNode;

   public Form1()
   {
       InitializeComponent();
       splitter1.BackColor = Color.Red;
       DriveTreeInit();
   }

   public void DriveTreeInit()
   {
       string[] drivesArray = Directory.GetLogicalDrives();
       treeView1.BeginUpdate();
       treeView1.Nodes.Clear();
       foreach (string s in drivesArray)
       {
           TreeNode drive = new TreeNode(s, 0, 0);
           treeView1.Nodes.Add(drive);
           GetDirs(drive);
       }
       treeView1.EndUpdate();
   }

   /// Получение списка каталогов
   public void GetDirs(TreeNode node)
   {
       DirectoryInfo[] diArray;
       node.Nodes.Clear();
       string fullPath = node.FullPath;
       DirectoryInfo di = new DirectoryInfo(fullPath);
       try
       {
         diArray = di.GetDirectories();
       }
       catch
       {
           return;
       }

       foreach (DirectoryInfo dirinfo in diArray)
       {
           TreeNode dir = new TreeNode(dirinfo.Name, 0, 0);
           node.Nodes.Add(dir);
       }
   }

  private void treeView1_BeforeExpand(object sender, TreeViewCancelEventArgs
e)
   {
       treeView1.BeginUpdate();
       foreach (TreeNode node in e.Node.Nodes)
       {
           GetDirs(node);
       }
```

```
            treeView1.EndUpdate();
        }

        private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
        {
            selectedNode = e.Node;
            tnSelect = e.Node;
            panel1.Invalidate();
        }

        private void panel1_Paint(object sender, PaintEventArgs e)
        {
             Panel panel = (Panel)sender;
             Graphics gr = e.Graphics;
             string fullPath = selectedNode.FullPath;
             DirectoryInfo di = new DirectoryInfo(fullPath);
             FileInfo[] fiArray;
             DirectoryInfo[] diArray;
             try
             {
                 fiArray = di.GetFiles();
                 diArray = di.GetDirectories();
             }
             catch
             {
                 return;
             }
             float y = 0;
             float d = (float)(Font.GetHeight()*1.3);
             foreach (DirectoryInfo dirInfo in diArray)
             {
        gr.DrawString(dirInfo.Name.ToString(), Font, new SolidBrush(Color.Blue), 5,
y);
                 y += d;
             }

            foreach (FileInfo fileInfo in fiArray)
             {
             gr.DrawString(fileInfo.Name.ToString(),
              Font, new SolidBrush(panel.ForeColor), 15, y);
                 y += d;
             }
            this.AutoScrollMinSize = new Size(0,(int)y);
        }
    }
}
```

Результат виконання програми наведено на рис. 4.64.

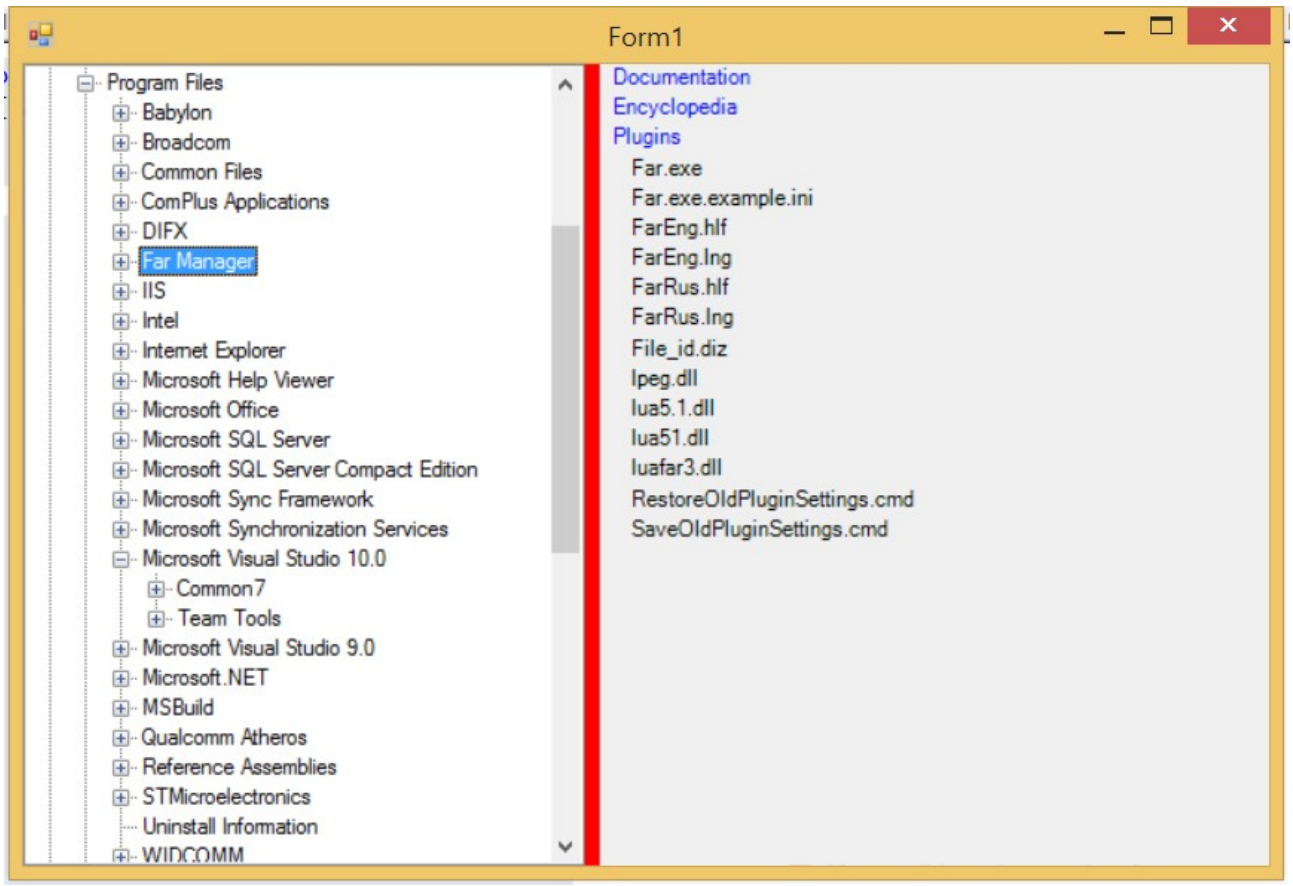


Рис. 4.64. До використання елемента *TreeView*

4.17. Використання фреймів

Для реалізації фреймів нам знадобиться елемент керування *Splitter*, який використовується як роздільник вікна на окремі фрейми. При цьому надається можливість регулювання розмірів вікон фреймів. Цей елемент був використаний у попередньому розділі.

Спеціально для демонстрації способів створення програм з фреймами, ми розробимо програму, в якій створюються три фрейми, що включають відповідно такі елементи керування:

1. TreeView
2. ListView
3. RichTextBox

Ці елементи мають бути інформаційно пов’язані. Послідовність роботи програми ось у чому. У *TreeView* вибирається папка. Її вміст (файли та підкаталоги) виводиться в *ListView*. При "кликанні" текстового файлу його вміст виводиться в *RichTextBox*.

Поділ вікна на три фрейми доцільно робити в режимі проектування, тобто у дизайнері. Розділимо основне вікно на три фрейми як це показано на рис. 4.65. Лівий містить *TreeView*, верхній правий - *ListView*, нижній правий - *RichTextBox*. Створення таких кадрів виконується у наступній послідовності.

Додамо у вікно нашої програми елемент керування *TreeView*, перетягнувши мишею його значок з панелі інструментів *Toolbox* у вікно форми. Створюється об’єкт *TreeView1*. Встановимо у вікні редагування властивості *Dock* елемента *TreeView1* значення, що дорівнює *Left* (рис. 4.66). Забезпечимо елемент керування *TreeView* роздільником. Для цього перетягнемо піктограму елемента керування *Splitter* з панелі інструментів *Toolbox* у вікно форми. Розділювач буде автоматично розташований праворуч від вікна елемента керування *TreeView*. Потім перетягуємо у вікно форми значок *ListView*. Створюється об’єкт *ListView1*, якому встановлюється значення властивості *Dock*, що дорівнює *Top*. Після цього встановлюємо роздільник. Остання дія - додавання елемента керування *RichTextBox*. Перетягнемо його значок з панелі *Toolbox* і встановимо значення властивості *Dock*, що дорівнює *Fill*.

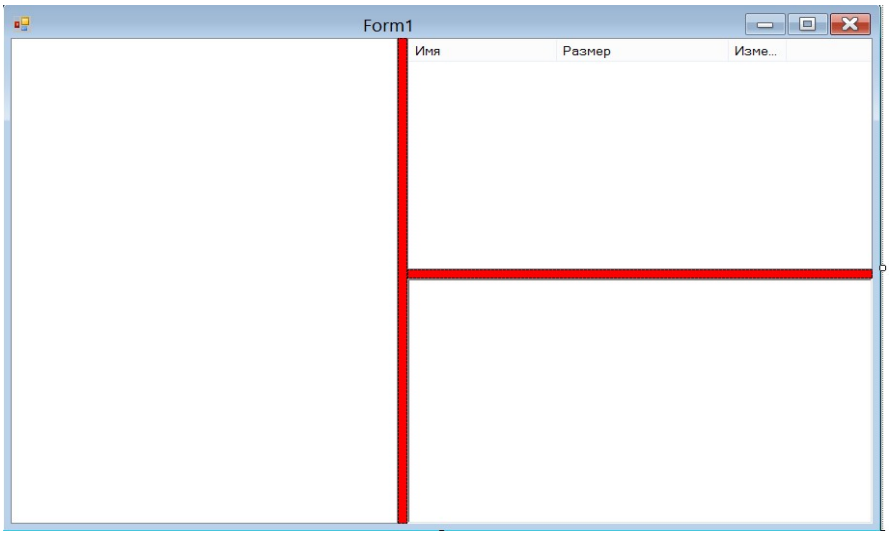


Рис. 4.65. Встановлення фреймів вікна

Рис. 4.66. Встановлення властивості Dock TreeView

У попередньому прикладі програмувалося дерево без встановлення значків у вузлах. У нашому випадку доповнимо дерево із встановленням трьох значків: зображення для дисків, зображення закритої папки, зображення для відкритої папки. З цією метою перетягнемо з панелі *Toolbox*

у вікно значок компонента *ImageList* (об'єкт *imageList1*). Заповнення цього елемента наводилося при вивченні *Listview*. При цьому використано папки значків (для *Windows 8.1* та *Studio 10*):

Program Files (x86)\Microsoft Visual Studio\Common\Graphics\Bitmaps\Outline\NoMask.
Program Files (x86)\Microsoft Visual Studio\Common\Graphics\Bitmaps\Outline\RedMask.

Копіювання файлів виконано при натисканні кнопки *Add* у вікні *Image Collection Editor*, показаного на рис. 4.67. Нагадаємо, що це вікно відкривається при виборі властивостей *imageList1* та виділенні властивості *Images*.

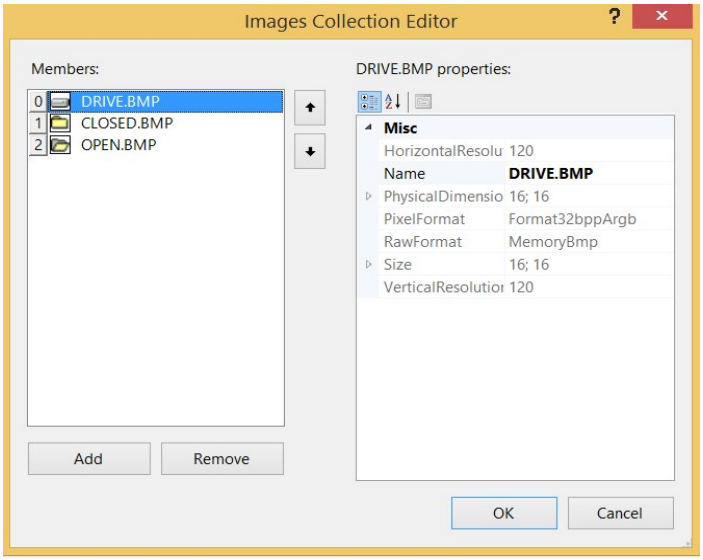


Рис. 4.67. Додавання значків диска та папок

Для підключення списку зображень до дерева властивості *ImageList* елемента керування *treeView1* надається значення *ImageList1*.

Надавши властивості *CheckBoxes* значення *True*, можна забезпечити вузли дерева індивідуальними прапорцями. Позначивши всі або деякі вузли дерева прапорцями, можна виконувати групові операції над відповідними об'єктами. Якщо дерево показує структуру дисків і файлів, це можуть бути такі операції, як, наприклад, видалення, переміщення чи копіювання.

Змінивши значення властивості *LabelEdit* на *True*, можна дозволити користувачам редагувати текстові написи, розташовані біля вузлів дерева.

Якщо дерево пов'язано, наприклад, з файловою системою, то така можливість стане в нагоді для перейменування папок.

Властивість *Sorting*, що задає сортування списку, може мати значення *None*, *Ascending* та *Descending*. У першому випадку список відображає елементи в порядку, в якому вони були додані. Значення *Ascending* визначає сортування в порядку зростання, а значення *Descending* — в порядку зменшення.

Оскільки ми плануємо відображати у правому верхньому фреймі елемент керування *ListView*, що включає значки. Необхідно додати ще один список зображень *ImageList*. Перетягніть його значок з панелі *Toolbar* у вікно дизайнера форм. Новий список матиме ідентифікатор *imageList2*.

У нашому додатку ми будемо використовувати 6 значків для відображення папок та файлів різних типів (файли програм, файлів довідкової системи, текстових файлів, файлів документів тощо). Опис завантаження зображень *imageList2* повністю збігається з описом такої операції у розділі 4.15. "ListView". Нагадаємо, що файли списку розташовано в наступному порядку, показаному на рис. 4.68.

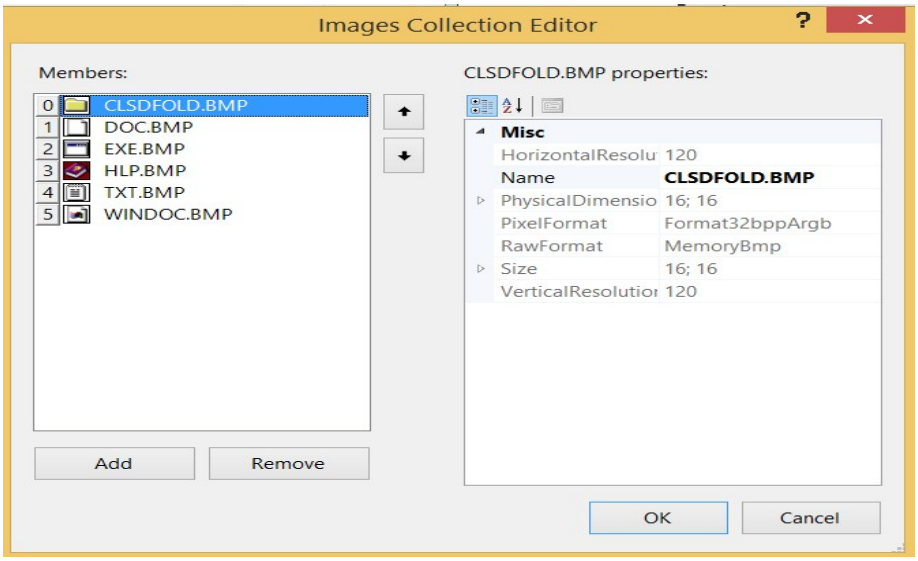


Рис. 4.68. До формування списку зображень у imageList2

У нашому додатку список *ListView* відображається як детальна таблиця. І тут властивість *View* має значення *Details*. Для підключення значків невеликого розміру до *ListView* встановлюємо властивості *SmallImageList* значення *imageList2*

У режимі дизайнера ми повністю підготували інформаційний базис для розробки програми. Слід зазначити, що значна частина коду вже розглядалася у розділах вивчення елементів керування *ListView* та *TreeView*. Тут уперше розглядається завантаження зображень у *Treeview*, наповнення списку *ListView*, використання елемента керування *RichTextBox* та відображення у ньому текстових файлів, виділених у *ListView*.

Для відображення значків у вузлах дерева нам необхідно змінити вихідний текст методів *DriveTreeInit* та *GetDirs*. Нагадаємо, що перший з цих методів ініціалізує дерево, а другий - додає до вузла дерева список каталогів.

Зверніть увагу на конструктор класу *TreeNode*, який створює вузли дерева всередині тіла циклу *foreach*:

```
public void DriveTreeInit()
{
    string[] drivesArray = Directory.GetLogicalDrives();
    treeView1.BeginUpdate();
    treeView1.Nodes.Clear();
    foreach (string s in drivesArray)
    {
        TreeNode drive = new TreeNode(s, 0, 0);
        treeView1.Nodes.Add(drive);
        GetDirs(drive);
```

```
        }
        treeView1.EndUpdate();
    }
```

Цей конструктор має три параметри. Про перший параметр ми вже розповідали – він встановлює текст напису для вузла дерева. Тепер настав час розповісти і про два інших параметри.

Якщо до елемента керування *TreeView* підключено список зображень, то другий та третій параметри конструктора класу *TreeNode* задають індекси зображень для вузла дерева. При цьому другий параметр визначає зображення невиділеного вузла дерева, а третій виділеного.

Що стосується методу *DriveTreeInit*, то розташований у ньому конструктор створює вузли, що відображають лише дискові пристрої. У будь-якому стані (як виділеному, так і невиділеному) нам необхідно відображати один і той же значок дискового пристрою, що має в нашому випадку індекс 0. Тому другий і третій параметри конструктора передають нульові значення.

Інша справа - метод *GetDirs*:

```
public void GetDirs(TreeNode node)
{
DirectoryInfo[] diArray;
node.Nodes.Clear();
string fullPath = node.FullPath;
DirectoryInfo di = new DirectoryInfo(fullPath);

try
{
diArray = di.GetDirectories();
}
catch
{
return;
}

foreach (DirectoryInfo dirinfo in diArray)
{
TreeNode dir = new TreeNode(dirinfo.Name, 1, 2);
node.Nodes.Add(dir);
}
}
```

Тут конструктору класу *TreeNode*, розміщеному всередині оператора циклу *foreach*, через другий та третій параметри ми передаємо індекси значків закритої та відкритої папки, відповідно. У результаті відображення дерева, папки, які виділив користувач, виділяються своїм позначенням (рис. 4.69).

Рис. 4.69. Дерево зі значками дискових пристроїв та папок

Тепер нашим завданням буде наповнити функціональністю список *ListView*. Коли користувач виділяє диск або каталог у дереві *TreeView*, створюється подія *AfterSelect*. Оброблювач цієї події повинен визначити, який диск або який каталог був виділений, а потім наповнити вікно списку *ListView* іменами каталогів і файлів, які розташовані на цьому диску або в цьому каталозі.

Створіть обробник події *AfterSelect* у такому вигляді, як показано нижче:

```
private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)

        {
            TreeNode selectedNode = e.Node;

                selectedNode = e.Node;
                tnSelect = e.Node;
                fullPath = selectedNode.FullPath;
                //MessageBox.Show(fullPath);
            DirectoryInfo di = new DirectoryInfo(fullPath);
            FileInfo[] fiArray;
            DirectoryInfo[] diArray;

            try
            {
                fiArray = di.GetFiles();
                diArray = di.GetDirectories();
            }
            catch
            {
                return;
            }

            listView1.Items.Clear();

            foreach (DirectoryInfo dirInfo in diArray)
            {
                ListViewItem lvi = new ListViewItem(dirInfo.Name);
                lvi.SubItems.Add("");
                lvi.SubItems.Add(dirInfo.LastWriteTime.ToString());
                lvi.ImageIndex = 0;
```

```
            listView1.Items.Add(lvi);
        }

        foreach (FileInfo fileInfo in fiArray)
        {
            ListViewItem lvi = new ListViewItem(fileInfo.Name);
            lvi.SubItems.Add(fileInfo.Length.ToString());
            lvi.SubItems.Add(fileInfo.LastWriteTime.ToString());
            string filenameExtension =
              Path.GetExtension(fileInfo.Name).ToLower();
            switch (filenameExtension)
            {
                case ".com":
                    {
                        lvi.ImageIndex =  2;
                        break;
                    }
                case ".exe":
                    {
                        lvi.ImageIndex =  2;
                        break;
                    }
                case ".hlp":
                    {
                        lvi.ImageIndex = 3;
                        break;
                    }
                case ".txt":
                    {
                        lvi.ImageIndex = 4;
                        break;
                    }
                case ".doc":
                    {
                        lvi.ImageIndex = 5;
                        break;
                    }
                default:
                    {
                        lvi.ImageIndex = 1;
                        break;
                    }
            }

            listView1.Items.Add(lvi);
        }
    }
```

Розповімо про те, як працює цей обробник подій.

Перш за все, метод отримує посилання на вузол дерева, виділений користувачем, з властивості *Node* параметра обробника подій *treeView1_OnAfterSelect*.

```
TreeNode selectedNode = e.Node;
```

Далі, повний шлях до виділеного вузла записується в полі *fullPath* класу *string* (Вам необхідно створити таке поле у класі *Form1*):

```
fullPath = selectedNode.FullPath;
```

Так як наше дерево містить лише позначення дисків і каталогів, то це буде шлях до кореневого каталогу диска, або до одного з підкаталогів.

Далі ми створюємо об'єкт класу *DirectoryInfo* і отримуємо списки всіх файлів і каталогів, які розміщені в каталозі, виділеному користувачем у дереві:

```
DirectoryInfo di = new DirectoryInfo(fullPath);
FileInfo[] fiArray;
DirectoryInfo[] diArray;
try
{
    fiArray = di.GetFiles();
    diArray = di.GetDirectories();
}
catch
{
    return;
}
```

Для виконання цих операцій застосовуються методи *GetFiles* та *GetDirectories*. Список файлів обробник події зберігає в масиві *fiArray*, а список каталогів - в масиві *diArray*.

Озброївшись переліками файлів і каталогів, ми почнемо додавати елементи до списку *ListView*, попередньо очистивши вміст списку методом *Clear*:

```
listView1.Items.Clear();
```

Наповнення списку іменами каталогів виконується в циклі *foreach*:

```
foreach (DirectoryInfo dirInfo in diArray)
{
    ListViewItem lvi = new ListViewItem(dirInfo.Name);
    lvi.SubItems.Add("");
    lvi.SubItems.Add(dirInfo.LastWriteTime.ToString());
    lvi.ImageIndex = 0;

    listView1.Items.Add(lvi);
}
```

Всі дії, що тут виконуються, було описано раніше. За допомогою конструктора класу *ListViewItem* ми створюємо елемент списку, а потім встановлюємо значення атрибутів цього елемента. Довжина каталогів вважається рівною нулю, а час останньої зміни каталогу отримується за допомогою методу *LastWriteTime* і перетворюється на текстовий рядок методом *ToString*.

У властивості *lvi.ImageIndex* записується нульове значення - індекс значка у списку зображень *imageList2* із зображенням закритої папки.

Після заповнення всіх атрибутів елемента списку, цей елемент додається до списку методом *listView1.Items.Add*.

На наступному етапі до списку додаються імена файлів, які розташовані у виділеному каталозі. Ці імена також додаються в циклі:

```
foreach (FileInfo fileInfo in fiArray)
{
    ListViewItem lvi = new ListViewItem(fileInfo.Name);
    lvi.SubItems.Add(fileInfo.Length.ToString());
    lvi.SubItems.Add(fileInfo.LastWriteTime.ToString());

    string filenameExtension =
      Path.GetExtension(fileInfo.Name).ToLower();

    switch (filenameExtension)
    {
        case ".com":
            {
                lvi.ImageIndex =  2;
                break;
            }
        case ".exe":
            {
                lvi.ImageIndex =  2;
                break;
            }
        case ".hlp":
            {
                lvi.ImageIndex = 3;
                break;
            }
        case ".txt":
            {
                lvi.ImageIndex = 4;
                break;
            }
        case ".doc":
            {
                lvi.ImageIndex = 5;
                break;
            }
        default:
            {
                lvi.ImageIndex = 1;
                break;
            }
    }
          listView1.Items.Add(lvi);
}
```

Розмір чергового файлу ми отримуємо за допомогою властивості *Length*, а дату останньої зміни – за допомогою властивості *LastWriteTime* (як і для каталогів).

Що ж до значків, то тут алгоритм трохи складніший. Нам потрібно визначити тип поточного файлу, проаналізувавши розширення його імені, а потім вибрати та записати у властивість *lvi.ImageIndex* індекс відповідного значка. Розширення імені файлу витягується з повного імені методом *Path.GetExtension*, а потім усі його літери перетворюються на маленькі методом *ToLower*. Безпосередній вибір піктограми здійснюється за допомогою оператора *switch*.

Підготовлений елемент списку, який описує поточний файл, додається до списку методом *listView1.Items.Add*.

Отже, тепер наша програма може відображати у лівому фреймі дерево каталогів, а у правому верхньому фреймі - вміст обраних каталогів. Залишився незадіяним лише правий нижній фрейм, у якому розташовано вікно текстового редактора *RichTextBox*.

Передбачивши обробник події *SelectedIndexChanged* до елемента керування *ListView*, ми зможемо відображати у вікні текстового редактора вміст текстових файлів, що відображаються у верхньому правому фреймі. Ця подія виникає, коли користувач двічі натискає по рядку у вікні списку *ListView*.

Ось вихідний текст нашого обробника події *SelectedIndexChanged*:

```
private void listView1_SelectedIndexChanged(object sender, EventArgs e)
   {

     foreach (ListViewItem lvi in listView1.SelectedItems)
         {
             string ext = Path.GetExtension(lvi.Text).ToLower();
             if (ext == ".txt" || ext == ".htm" || ext == ".html")
             {
                 try
                 {
                     richTextBox1.LoadFile(Path.Combine(fullPath, lvi.Text),
                     RichTextBoxStreamType.PlainText);
                     sb.Text = lvi.Text;
                     richTextBox1.Clear();
                 }
                 catch
                 {
                     return;
                 }
             }
             else if (ext == ".rtf")
             {
                 try
                 {
                   richTextBox1.LoadFile(Path.Combine(fullPath, lvi.Text),
                   RichTextBoxStreamType.RichText);
                   sb.Text = lvi.Text;
                 }
                catch
                 {
                     return;
                 }
             }
         }
      }
   }
```

Отримавши керування, обробник подій, перш за все, повинен одержати список елементів, виділених користувачем у вікні списку. Ця операція виконується у циклі:

```
foreach(ListViewItem lvi in listView1.SelectedItems)
                              {
                              ...
                              }
```

Ідентифікатори виділених елементів зберігаються в наборі *listView1.SelectedItems*.

У загальному випадку користувач може виділити кілька елементів, проте якщо користувач натиснув по рядку двічі, то в результаті виявиться виділеним лише цей рядок. Тому наш цикл виконає лише одну ітерацію.

Оскільки редактор *RichTextBox* здатний відображати вміст лише текстових файлів, нам необхідно визначити тип файлу, вибраного користувачем. Ми робимо це, аналізуючи розширення імені файлу, отримане за допомогою методу *GetExtension* (з перетворенням символів розширення на символи нижнього регістру):

```
string ext = Path.GetExtension(lvi.Text).ToLower();
```

Якщо розширення імені відповідає текстовому файлу або файлу *HTML*, ми завантажуємо вміст файлу у вікно редактора *RichTextBox*, користуючись при цьому методом *LoadFile*:

```
foreach (ListViewItem lvi in listView1.SelectedItems)
    {
        string ext = Path.GetExtension(lvi.Text).ToLower();
        if (ext == ".txt" || ext == ".htm" || ext == ".html")
        {
            try
            {
                richTextBox1.LoadFile(Path.Combine(fullPath, lvi.Text),
                RichTextBoxStreamType.PlainText);
                sb.Text = lvi.Text;
                richTextBox1.Clear();
            }
            catch
            {
                return;
            }
        }
        else if (ext == ".rtf")
        {
            try
            {
                richTextBox1.LoadFile(Path.Combine(fullPath, lvi.Text),
                RichTextBoxStreamType.RichText);
                sb.Text = lvi.Text;
            }

            catch
            {
                return;
            }
        }
```

Першим параметром методу передається повний шлях до файлу, отриманий комбінуванням повного шляху каталогу *fullPath*, виділеного в дереві, та імені файлу *lvi.Text*, виділеного у списку *ListView*.

Через другий параметр методу *LoadFile* передається тип документа, що завантажується, заданий як *RichTextBoxStreamType.PlainText* (тобто текстовий документ).

Додатково ім'я документа відображається у рядку стану головного вікна програми, навіщо це ім'я переписується з властивості *lvi.Text* у властивість *statusBar1.Text*.

Обробка файлів із документами типу *RTF*, що мають розширення імені *rtf*, виконується аналогічно. При цьому тип документа вказується методом *LoadFile* як *RichTextBoxStreamType.RichText*.

Приклад 42.

Повний текст вихідного тексту наведено в лістингу 4.44.

На рис. 4.70 показаний зовнішній вигляд головного вікна програми, коли до нижнього правого фрейму завантажено документ *RTF*.

Лістинг 4.44.(examp42)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace examp42
{
    public partial class Form1 : Form
    {

        TreeNode tnSelect;
        StatusBar sb;
        string fullPath;
        public Form1()
        {
            sb = new StatusBar();
            sb.Parent = this;

            InitializeComponent();
            DriveTreeInit();
        }

        public void DriveTreeInit()
        {
            string[] drivesArray = Directory.GetLogicalDrives();
            treeView1.BeginUpdate();
            treeView1.Nodes.Clear();
            foreach (string s in drivesArray)
            {
                TreeNode drive = new TreeNode(s, 0, 0);
                treeView1.Nodes.Add(drive);
                GetDirs(drive);
            }
            treeView1.EndUpdate();
        }
```

```
/// Получение списка каталогов
public void GetDirs(TreeNode node)
{
    DirectoryInfo[] diArray;
    node.Nodes.Clear();
    string fullPath = node.FullPath;
    DirectoryInfo di = new DirectoryInfo(fullPath);
    try
    {
        diArray = di.GetDirectories();
    }
    catch
    {
        return;
    }
    foreach (DirectoryInfo dirinfo in diArray)
    {
        TreeNode dir = new TreeNode(dirinfo.Name, 1, 2);
        node.Nodes.Add(dir);
    }
}
private void treeView1_BeforeExpand(object sender,
TreeViewCancelEventArgs e)
{
    treeView1.BeginUpdate();
    foreach (TreeNode node in e.Node.Nodes)
    {
        GetDirs(node);
    }
    treeView1.EndUpdate();
}
private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
{
    TreeNode selectedNode = e.Node;
        selectedNode = e.Node;
        tnSelect = e.Node;
        fullPath = selectedNode.FullPath;
        //MessageBox.Show(fullPath);
    DirectoryInfo di = new DirectoryInfo(fullPath);
    FileInfo[] fiArray;
    DirectoryInfo[] diArray;
    try
    {
        fiArray = di.GetFiles();
        diArray = di.GetDirectories();
    }
    catch
    {
        return;
    }
    listView1.Items.Clear();
    foreach (DirectoryInfo dirInfo in diArray)
    {
        ListViewItem lvi = new ListViewItem(dirInfo.Name);
        lvi.SubItems.Add("");
        lvi.SubItems.Add(dirInfo.LastWriteTime.ToString());
        lvi.ImageIndex = 0;
        listView1.Items.Add(lvi);
    }
    foreach (FileInfo fileInfo in fiArray)
    {
        ListViewItem lvi = new ListViewItem(fileInfo.Name);
```

```
            lvi.SubItems.Add(fileInfo.Length.ToString());
            lvi.SubItems.Add(fileInfo.LastWriteTime.ToString());
            string filenameExtension =
              Path.GetExtension(fileInfo.Name).ToLower();
            switch (filenameExtension)
            {
                case ".com":
                    {
                        lvi.ImageIndex =  2;
                        break;
                    }
                case ".exe":
                    {
                        lvi.ImageIndex =  2;
                        break;
                    }
                case ".hlp":
                    {
                        lvi.ImageIndex = 3;
                        break;
                    }
                case ".txt":
                    {
                        lvi.ImageIndex = 4;
                        break;
                    }
                case ".doc":
                    {
                        lvi.ImageIndex = 5;
                        break;
                    }
                default:
                    {
                        lvi.ImageIndex = 1;
                        break;
                    }
            }
            listView1.Items.Add(lvi);
        }
    }

   private void listView1_SelectedIndexChanged(object sender, EventArgs e)
{
  foreach (ListViewItem lvi in listView1.SelectedItems)
      {
          string ext = Path.GetExtension(lvi.Text).ToLower();
          if (ext == ".txt" || ext == ".htm" || ext == ".html")
          {
              try
              {
                  richTextBox1.LoadFile(Path.Combine(fullPath, lvi.Text),
                  RichTextBoxStreamType.PlainText);
                  sb.Text = lvi.Text;
                  richTextBox1.Clear();
              }
              catch
              {
                return;
              }
          }
          else if (ext == ".rtf")
          {
```

```
                    try
                    {
                        richTextBox1.LoadFile(Path.Combine(fullPath, lvi.Text),
                         RichTextBoxStreamType.RichText);
                        sb.Text = lvi.Text;
                    }
                    catch
                    {
                        return;
                    }
                }
            }
        }
    }
}
```

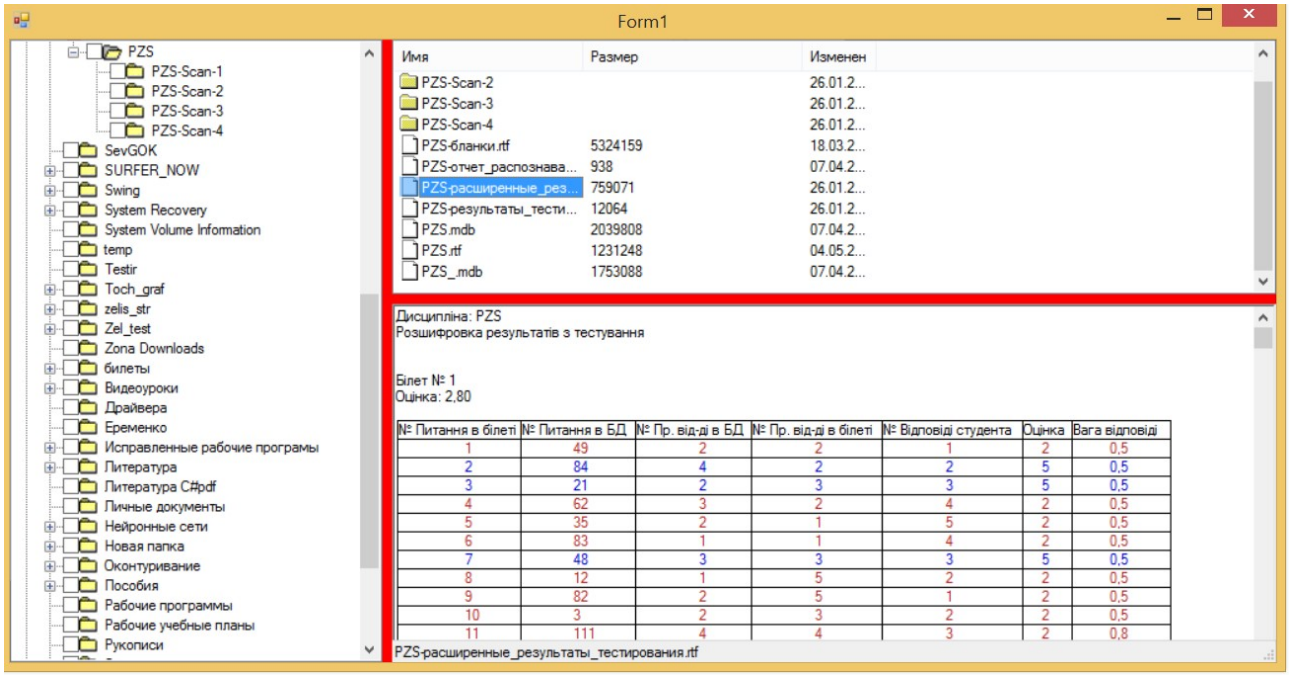


Рис. 4.70. Перегляд документа RTF у вікні нашого додатку

4.18. Елемент керування TabControl

Елемент керування *Windows Forms TabControl* використовується для відображення декількох вкладок, аналогічних роздільникам у записнику або міткам у наборі папок картотечного блоку. Вкладки можуть містити малюнки та інші елементи керування. Елемент керування вкладками можна використовувати для створення багатосторінкових діалогових вікон, на кшталт тих, що відображаються у багатьох компонентах інтерфейсу операційної системи *Windows*, наприклад, у властивостях екрана панелі керування. Крім того, *TabControl* можна використовувати для створення сторінок властивостей для групування пов'язаних властивостей.

Найбільш важливою властивістю елемента керування *TabControl* є властивість *TabPages*, що містить окремі вкладки. Кожна вкладка є об'єктом *TabPage*. Якщо перейти на вкладку, подія *Click* викликається для відповідного об'єкта *TabPage*.

Зовнішній вигляд вкладок у формах *Windows Forms* може бути змінений за допомогою властивостей елемента керування *TabControl* та об'єктів *TabPage*, які представляють окремі вкладки у складі елемента керування. Налаштування цих властивостей дозволяє відображати малюнки на вкладках, розміщувати вкладки вертикально, розміщувати їх у кілька рядів, а також включати та вимикати вкладки програмними засобами.

Для відображення значка в області написів на вкладці необхідно:

1. Додати елемент керування *ImageList* у форму.
2. Додати зображення до списку зображень.
3. Задати для властивості *ImageList* елемента керування *TabControl* значення *ImageList*.
4. Привласнити властивості *ImageIndex* об'єкта *TabPage* значення індексу відповідного зображення зі списку.

Для створення кількох рядів вкладок необхідно:

1. Додати потрібну кількість сторінок вкладок.
2. Привласнити властивості *Multiline* елементу *TabControl* значення *true*.
3. Якщо вкладки не відображаються в кілька рядів, зменшити значення властивості *Width* елемента керування *TabControl* так, щоб воно поменшало загальної ширини всіх вкладок.

Для розташування вкладок уздовж краю елемента керування необхідно надати властивості *Alignment* об'єкта *TabControl* значення *Left* або *Right*.

Для відображення вкладок у вигляді кнопок надайте властивості *Appearance* об'єкта *TabControl* значення *Buttons* або *FlatButtons*.

За замовчуванням елемент керування *TabControl* містить два елементи керування *TabPage*. Доступ до цих вкладок можливий за допомогою *TabPages*.

Для додавання вкладки програмними засобами використовується метод *Add* властивості *TabPages*:

```
string title = "TabPage " + (tabControl1.TabCount + 1).ToString();
TabPage myTabPage = new TabPage(title);
tabControl1.TabPages.Add(myTabPage);
```

Щоб видалити вибрані вкладки, використовуйте метод *Remove* властивості *TabPages*.

Щоб видалити всі вкладки, використовуйте метод *Clear* властивості *TabPages*. Приклади:

```
tabControl1.TabPages.Remove(tabControl1.SelectedTab);
tabControl1.TabPages.Clear();
```

Поставимо завдання написати програму, що дозволяє вибрати текст з двох варіантів, задати колір та розмір шрифту цього тексту на трьох вкладках *TabControl* з використанням перемикачів *RadioButton*. Фрагмент роботи програми наведено на рис. 4.71.

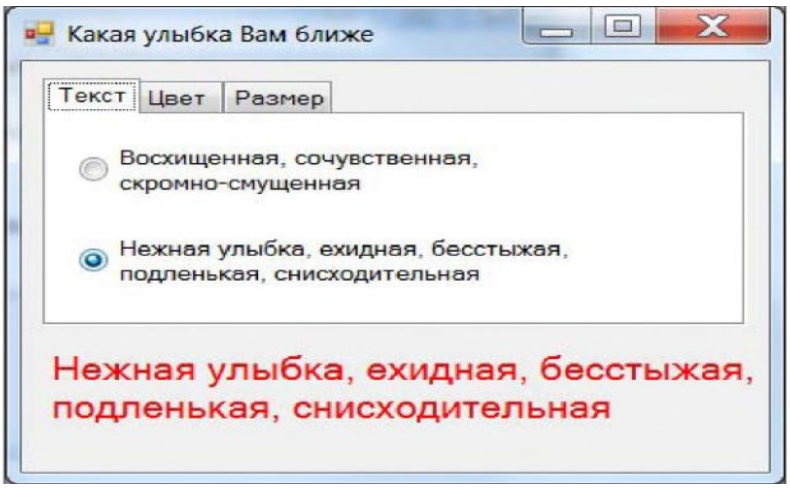


Рис. 4.71. До використання елемента керування *TabControl*

Використовуючи панель елементів *Toolbox*, перетягніть у форму мишею елемент керування *TabControl*. За замовчуванням передбачено дві вкладки, а за умовою завдання нам потрібні три. Додати третю вкладку можна в конструкторі форми та програмно.

Спочатку покажемо, як можна додати третю вкладку у конструкторі. Для цього у властивостях (вікно *Proprties*) елемента керування *TabControl* вибираємо властивість *TabPage* та в результаті потрапляємо у діалогове вікно *TabPage Collection*, де додаємо (кнопка *Add*) третю кнопку. Ці вкладки нумеруються з нуля, тобто вкладка визначається як *TabPages* (2). Назва кожної вкладки використовується у програмі.

Для більшої глибини знання покажемо додавання третьої кнопки в програмному режимі. Перед програмуванням для кожної вкладки вибираємо з панелі елементів *Toolbox* два перемикачі *RadioButton*, а до форми перетягуємо мітку *Label*.

Приклад 43.

Програмний код наведено в лістингу 4.45.

Лістинг 4.45. (examp43)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp43
{
    public partial class Form1 : Form
    {

        private TabPage tabPage3;
        private RadioButton radioButton5;
        private RadioButton radioButton6;
```

```
        public Form1()
        {
            InitializeComponent();
            AutoSize = true;
            tabPage3 = new TabPage();
            tabPage3.Location = new System.Drawing.Point(4, 25);
           // tabPage3.Name = "tabPage3";
            tabPage3.Padding = new System.Windows.Forms.Padding(3);
            tabPage3.Size = new System.Drawing.Size(342, 251);
            tabPage3.TabIndex = 2;
            tabPage3.Text = "tabPage3";
            tabPage3.UseVisualStyleBackColor = true;
// ------------------------------------------------------------------
            radioButton5 = new RadioButton();
            radioButton5.AutoSize = true;
            radioButton5.Location = new System.Drawing.Point(50, 50);
            radioButton5.Name = "radioButton5";
            radioButton5.Text = "";
            radioButton5.UseVisualStyleBackColor = true;
            radioButton5.Click += new System.EventHandler(radioButton_Click);

            radioButton6 = new RadioButton();
            radioButton6.AutoSize = true;
            radioButton6.Location = new System.Drawing.Point(50, 120);
            radioButton6.Name = "radioButton6";
            radioButton6.Text = "";
            radioButton6.UseVisualStyleBackColor = true;
            radioButton6.Click += new System.EventHandler(radioButton_Click);
      // ------------------------------------------------------------------
            tabPage3.Controls.Add(radioButton5);
            tabPage3.Controls.Add(radioButton6);
            tabControl1.Controls.Add(tabPage3);
            Text = "Какая улыбка Вам ближе";
            tabControl1.TabPages[0].Text  = "Текст";
            tabControl1.TabPages[1].Text  = "Цвет";
            tabControl1.TabPages[2].Text =  "Размер";
            radioButton1.Text = "Восхищенная,сочувственная, " + "\n" +
            " скромно-смущенная";
            radioButton2.Text = "Нежная улыбка , ехидная, бестыжая, "+
            Environment.NewLine +           " подлинькая, снисходительная";

                radioButton3.Text = "Красный";
                radioButton4.Text = "Синий";
                radioButton5.Text = "10 пункт";
                radioButton6.Text = "15 пункт";
                label1.Font = new Font(label1.Name, 10);
                label1.Text = radioButton1.Text;
                radioButton1.Checked = true;
                radioButton5.Checked = true;
        }
        private void radioButton_Click(object sender, EventArgs e)
        {
            string st  = ((RadioButton)sender).Name;
            if (st.EndsWith("1")) label1.Text = radioButton1.Text;
            if (st.EndsWith("2")) label1.Text = radioButton2.Text;
            if (st.EndsWith("3")) label1.ForeColor = Color.Red;
            if (st.EndsWith("4")) label1.ForeColor = Color.Blue;
            if (st.EndsWith("5")) label1.Font = new Font(label1.Name, 10);
            if (st.EndsWith("6")) label1.Font = new Font(label1.Name, 15);
          }
    }
}
```

4.19. Елемент ErrorProvider

ErrorProvider - не елемент керування, а компонент. Коли ви перетягуєте компонент до дизайнера форм, він відображається в лотку компонентів під дизайнером. Призначення *ErrorProvider* полягає в тому, щоб висвічувати піктограму поруч із елементом керування, коли виникає помилкова ситуація або не відбувається перевірка. Припустимо, що ви маєте поле *TextBox*, призначене для введення віку. Скажімо, значення віку не повинно перевищувати 65. Якщо користувач намагатиметься ввести більше значення, його потрібно буде інформувати, що введено вік, що перевищує допустимий, і це слід виправити. Перевірка правильності введеного значення виконується в обробнику події *Validated* цього текстового поля. Якщо перевірка не пройшла, можна викликати метод *SetError*, передавши посилання на елемент керування, який викликав помилку, і коли користувач наведе курсор миші на піктограму, буде відображено текст повідомлення про помилку. На рис. 4.72 показано піктограму, яка з'являється у разі введення в текстове поле неприпустимого значення.

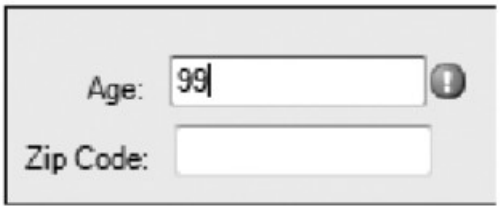


Рис. 4.72. Піктограма, котра з'являється у випадку введення
в текстове поле некоректного значення

Ви можете створити *ErrorProvider* для кожного елемента керування на формі, який може бути причиною помилки, але якщо у вас дуже багато елементів керування, це може виявитися занадто громіздко. Інший варіант - використовувати один постачальник помилок, і в події перевірки викликати метод *IconLocation* з тим елементом керування, який викликав перевірку, і одним із значень перерахування *ErrorIconAlignment*. Це значення встановлює вирівнювання піктограми елемента керування. Потім слід викликати метод *SetError*. Якщо немає помилкових умов, можна очистити *ErrorProvider*, викликавши *SetError* з порожнім рядком помилки. У наведеному нижче прикладі показано, як це працює.

```
private void txtAge_Validating(object sender,
Sytem.ComponentModel.CancelEventArgs e)
{
      if(txtAge.TextLength > 0 && Convert.ToInt32(txtAge.Text) > 65)
      {
            errMain.SetIconAlignment((Control)sender,
          ErrorIconAlignment.MiddleRight);
            errMain.SetError((Control)sender, "Значение должно быть меньше
          65.");
            e.Cancel = true;
      }
```

```
            else
            {
                 errMain.SetError((Control)sender, "");
            }
}

private void txtZipCode_Validating(object sender, CancelEventArgs e)
{
            if(txtZipCode.TextLength != 5)
            {
               errMain.SetIconAlignment((Control)sender,
           ErrorIconAlignment.MiddleRight);
               errMain.SetError((Control)sender, "Должно быть 5 символов.");
               e.Cancel = true;
            }
            else
            {
               errMain.SetError((Control)sender, "");
            }
}
```

Якщо перевірка не проходить (наприклад, у *txtAge* введено число більше 65), викликається метод *SetIcon* постачальника помилок *errMain*. Він встановлює піктограму поруч із елементом керування, який не пройшов перевірку. Також, тут встановлюється текст помилки, тому коли користувач наведе курсор миші на цю піктограму, побачить повідомлення, що інформує його про причину невдалої перевірки. Для другого *TextBox* (ім'я *txtZipCode*) кількість символів у тексті повинна дорівнювати 5.

Якщо користувач ввів понад п'ять символів, з'являється невелика піктограма помилки (!) поруч з другим *TextBox*. Якщо користувач підведе курсор миші до цієї піктограми, з'явиться "випливаючий" текст з описом помилки. *ErrorProvider* налаштований так, щоб змусити піктограму "блимати", що посилить візуальний вплив (звичайно, без запуску програми ви цього не побачите).

Якщо використовувати такий вид перевірки введення, необхідно вивчити деякі властивості класу *Control*, описи яких наводяться в табл. 4.5. *Control* це базовий клас для елементів керування, що є компонентами з візуальним поданням.

Таблиця 4.5

Властивості та події Control, які використовуються при контролю даних

Властивість або подія	**Призначення**
CausesValidation	Індикатор того, що вибір цього елемента керування викликає перевірку введення для елементів керування, які потребують такої перевірки
Validated	Подія, що генерується тоді, коли елемент керування закінчує виконання програмної логіки перевірки введення
Validating	Подія, що генерується тоді, коли елемент керування перевіряє введення користувача (наприклад, коли елемент керування втрачає фокус введення)

Кожен елемент графічного інтерфейсу може встановити для властивості *CausesValidation* значення *true* (істина) або *false* (брехня), причому значенням за замовчуванням є *true*. Якщо ви встановите для вказаних даних значення *true*, цей елемент керування при отриманні ним фокуса введення змусить інші елементи керування у формі виконати перевірку введення. При отриманні фокусу введення перевіряючим елементом керування, генеруються події *Validating* і *Validated* для кожного елемента керування. У контексті обробника події *Validating* необхідно конфігурувати відповідний *ErrorProvider*. Також можна, але необов'язково, обробити подію *Validated*, щоб визначити, коли елемент керування закінчить цикл перевірки.

ErrorProvider пропонує дуже невеликий набір членів. Для нашого прикладу найважливішою є властивість *BlinkStyle*, якій можна надати будь-яке значення з переліку *ErrorBlinkStyle*. Описи цих значень даються у табл. 4.6.

Таблиця 4.6

Значення ErrorBlinkStyle

Значення	Опис
AlwaysBlink	Змушує піктограму помилки "блимати", коли помилка відображається вперше або коли елементу керування призначається рядок нової помилки, а піктограма помилки вже відображається
BlinkIfDifferentError	Змушує піктограму помилки "блимати", коли піктограма помилки вже відображається, але елементу керування призначається рядок нової помилки
NeverBlink	Індикатор того, що піктограма помилки ніколи не повинна "блимати"

Приклад 44.

Розглянемо повніший приклад використання *ErrorProvider*. На форму встановляємо два *TextBox* та *Button*. Це показано на рис. 4.73. У верхньому *TextBox* користувач може вводити як ціле число (мається на увазі вводиться текст у вигляді цифр без дробової частини), так і дійсне, з роздільником цілої та дробової частини числа. При цьому максимальне значення числа не повинно перевищувати 1000. У нижньому *TextBox* можна вводити будь-який текст, довжина якого дорівнює 5. Якщо користувач введе в *TextBox()* більше п'яти символів і *TextBox* втрачає фокус введення, відображається інформація, показана на рис. 4.73.

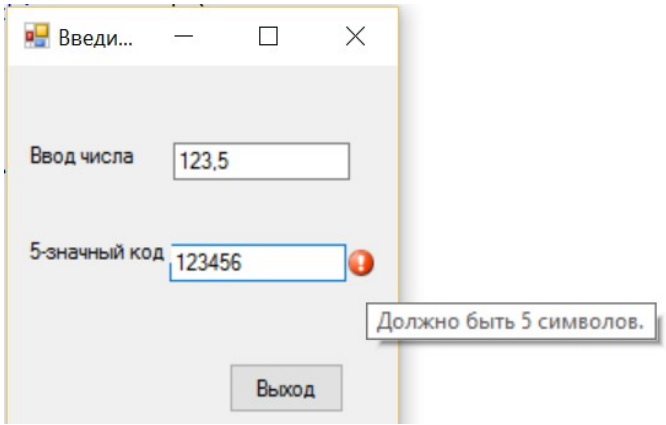


Рис. 4.73. Робота *ErrorProvider* у дії

У лістингу 4.46. наведено текст програми, який повністю проілюстровано. При цьому було максимально враховано різні фактори, які виникають під час введення інформації. Наведена програма досить універсальна. Тут при введенні даних у верхній *TextBox* навіть враховано роздільник (кома або точка) дробової та цілої частини числа.

Лістинг 4.46. (examp 44)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp44
{

     public partial class Form1 : Form
     {
        // Разделитель целой и дробной части числа может быть
        // точкой "." или запятой "," в зависимости от
        // установок в пункте Язык и региональные стандарты
        // Панели управления ОС Windows:
        System.Globalization.CultureInfo Культ = System.Globalization.
                                              CultureInfo.CurrentCulture;
        string ТчкИлиЗпт;
        public Form1()
        {
            InitializeComponent();
            this.Text = "Введите число";
            // Выясняем, что установлено на данном ПК в качестве
            // разделителя целой и дробной части: точка или запятая:
            ТчкИлиЗпт = Культ.NumberFormat.NumberDecimalSeparator;
        }

        private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (e.KeyChar == (char)Keys.Enter && textBox1.Text.Length > 0 )
            {
                if (textBox1.Text.Substring(textBox1.TextLength - 1) ==
ТчкИлиЗпт)
                { textBox1.Text = ""; return; }
                textBox2.Focus();
            }

            bool dr = true;
            contr_text(sender, e, dr, ТчкИлиЗпт);
        }

        private void contr_text(object sender, KeyPressEventArgs e, bool dr,
        string ТчкИлиЗпт)
        {
            bool ТчкИлиЗптНАЙДЕНА = false;
            // Разрешаю ввод десятичных цифр:
            if (char.IsDigit(e.KeyChar) == true) return;
            // Разрешаю ввод <BackSpace>:
            if (e.KeyChar == (char)Keys.Back) return;
            if (dr == true)
```

```
                {
                    // Поиск ТчкИлиЗпт в textBox, если IndexOf() == -1, то не
найдена:
                 if (textBox1.Text.IndexOf(ТчкИлиЗпт) != -1) ТчкИлиЗптНАЙДЕНА =
true;
                    // Если ТчкИлиЗпт уже есть в textBox, то запрещаем вводить и
ее,
                    // и любые другие символы:
                    if (ТчкИлиЗптНАЙДЕНА == true) { e.Handled = true; return; }
                    // Если ТчкИлиЗпт еще нет в textBox, то разрешаем ее ввод:
                    if (e.KeyChar.ToString() == ТчкИлиЗпт) return;
                    // В других случаях запрет на ввод:
                }
                    e.Handled = true;
            }

            private void textBox_Validating(object sender, CancelEventArgs e)
            {
               if (((Control)sender).Name == "textBox1")
               {
               if (textBox1.Text.Length > 0 && Convert.ToDouble(textBox1.Text) >
1000)
                   {
                       errorProvider1.SetIconAlignment((Control)sender,
                       ErrorIconAlignment.MiddleRight);
                       errorProvider1.SetError((Control)sender,
                       "Значение должно быть меньше 1000.");
                       e.Cancel = true;
                       return;
                   }
                   if (textBox1.Text.Length == 0)
                   {
                       errorProvider1.SetIconAlignment((Control)sender,
                       ErrorIconAlignment.MiddleRight);
                       errorProvider1.SetError((Control)sender, " Нет ввода ");
                       e.Cancel = true;
                       return;
                   }
               }
               if (((Control)sender).Name == "textBox2")
               {
                   if (textBox2.TextLength != 5)
                   {
                       errorProvider1.SetIconAlignment((Control)sender,
                       ErrorIconAlignment.MiddleRight);
                     errorProvider1.SetError((Control)sender, "Должно быть 5
символов.");
                       e.Cancel = true;
                       return;
                   }

                }
                       errorProvider1.SetError((Control)sender, "");
            }
            private void button1_Click(object sender, EventArgs e)
            {
                if (textBox1.TextLength == 0) textBox1.Focus();
                if (textBox2.TextLength == 0) textBox2.Focus();
                Close();
            }
        }
    }
```

4.20. Елемент HelpProvider

HelpProvider (постачальник довідки), подібно *ErrorProvider*, є компонентом, а не елементом керування. *HelpProvider* дозволяє зв'язати елементи керування з темами підказки. Щоб асоціювати елемент керування з постачальником довідки, необхідно викликати метод *SetShowHelp*, передавши елемент керування та булівське значення, що вказує на необхідність відображення тексту підказки. Властивість *HelpNamespace* дозволяє встановити файл довідки. Коли встановлено *HelpNamespace*, вміст довідкового файлу відображається в будь-який момент після натискання *<F1>*.

Приклад 45.

На рис. 4.74 показано приклад, у якому програмується довідка для трьох *TextBox*. Якщо натиснути кнопку ? (довідка) у рядку заголовка, а потім натиснути "мишкою" елемент керування, то з'являється підказка. При натисканні *F1* виводиться довідка, приклад якої наведено на рис. 4.75. Вміст довідки міститься у файлі-довідки "zeles.chm". Код програми наводиться у лістингу 4.47.

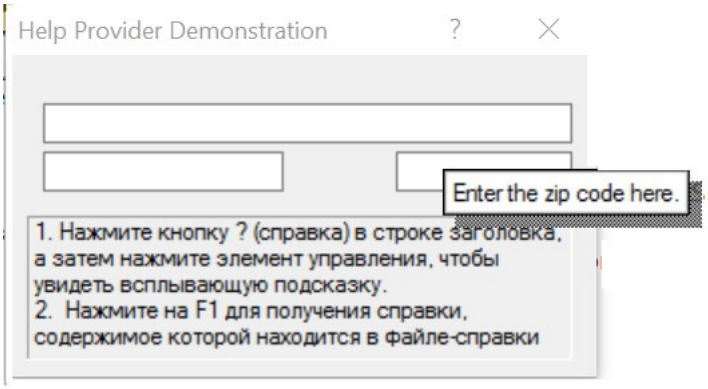


Рис. 4.74. До використання елемента HelpProvider

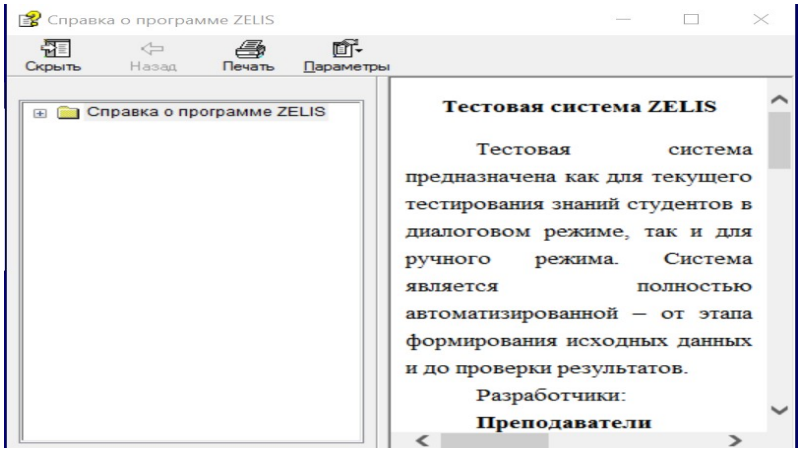


Рис. 4.75. Вміст довідкового файлу *.chm

Лістинг 4.47. (examp45)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp45
{
    public partial class Form1 : Form
    {
        private System.Windows.Forms.TextBox addressTextBox;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.TextBox cityTextBox;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.TextBox stateTextBox;
        private System.Windows.Forms.TextBox zipTextBox;
        private System.Windows.Forms.HelpProvider helpProvider1;
        private System.Windows.Forms.Label helpLabel;

        public Form1()
        {
            InitializeComponent();
            this.addressTextBox = new System.Windows.Forms.TextBox();
            this.helpLabel = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.cityTextBox = new System.Windows.Forms.TextBox();
            this.label3 = new System.Windows.Forms.Label();
            this.stateTextBox = new System.Windows.Forms.TextBox();
            this.zipTextBox = new System.Windows.Forms.TextBox();
            // Help Label
            this.helpLabel.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D;
            this.helpLabel.Location = new System.Drawing.Point(8, 80);
            this.helpLabel.Size = new System.Drawing.Size(272, 72);
            this.helpLabel.Text = "1. Нажмите кнопку ? (справка) в строке" +
            " заголовка, а затем" нажмите элемент" +
            " управления, чтобы увидеть всплывающую подсказку." +
            " \n2.  Нажмите на F1 для получения справки, содержимое которой" +
            "находится в файле-справки "
            " zeles.chm";
            this.label2.Location = new System.Drawing.Point(16, 8);
            this.label2.Size = new System.Drawing.Size(100, 16);
            this.label2.Text = "Address:";

            this.helpProvider1 = new System.Windows.Forms.HelpProvider();
            this.helpProvider1.SetShowHelp(this.addressTextBox, true);
            this.helpProvider1.SetHelpString(this.addressTextBox,
            "Enter the street" + "address in this text box.");
            this.helpProvider1.SetShowHelp(this.cityTextBox, true);
            this.helpProvider1.SetHelpString(this.cityTextBox,
            "Enter the city here.");
            this.helpProvider1.SetShowHelp(this.zipTextBox, true);
            this.helpProvider1.SetHelpString(this.zipTextBox,
            "Enter the zip code here.");
            this.helpProvider1.HelpNamespace = "zelis.chm";
            // Address TextBox
            this.addressTextBox.Location = new System.Drawing.Point(16, 24);
            this.addressTextBox.Size = new System.Drawing.Size(264, 20);
```

```
            this.addressTextBox.TabIndex = 0;
            this.addressTextBox.Text = "";

            this.cityTextBox.Location = new System.Drawing.Point(16, 48);
            this.cityTextBox.Size = new System.Drawing.Size(120, 20);
            this.cityTextBox.TabIndex = 3;
            this.cityTextBox.Text = "";

            this.zipTextBox.Location = new System.Drawing.Point(192, 48);
            this.zipTextBox.Size = new System.Drawing.Size(88, 20);
            this.zipTextBox.TabIndex = 6;
            this.zipTextBox.Text = "";

            this.Controls.AddRange(new System.Windows.Forms.Control[]
        {
            this.zipTextBox,
            this.label3, this.cityTextBox,
            this.label2, this.helpLabel,
            this.addressTextBox});
            this.FormBorderStyle =
            System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.HelpButton = true;
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.ClientSize = new System.Drawing.Size(292, 160);
            this.Text = "Help Provider Demonstration";
        }
    }
}
```

4.21. DateTimePicker та MonthCalendar

DateTimePicker

DateTimePicker дає можливість користувачу вибирати значення дати або часу (або того, й іншого) у безлічі різноманітних форматів. Можна відображати значення *DateTime* у будь-якому із стандартних форматів дати та часу. Властивість Format приймає значення типу переліку *DateTimePickerFormat*, які встановлюють формат *Long*, *Short*, *Time* або *Custom*. Якщо властивість *Format* встановлено в *DateTimePickerFormat.Custom*, то можна встановити властивості *CustomFormat* рядок, який представляє формат. Передбачено ще дві властивості - *Text* і *Value*. Властивість *Text* повертає текстове уявлення значень *DateTime*, тоді як *Value* повертає сам об’єкт типу *DateTime*. Можна також встановити максимальне та мінімальне допустиме значення за допомогою властивостей *MinDate* та *MaxDate*. Коли користувач натискає по стрілці, спрямованій вниз, відображається календар, який дозволяє вибрати потрібну дату.

Приклад 45.

У лістингу 4.48 та рис. 4.76 показаний приклад відображення календаря для вибору дати та відображення часу в текстовій частині елемента керування.

У дизайнері встановлено елемент керування *DateTimePicker* (ідентифікатор *dateTimePicker1*) та три *Label*.

Для полегшення пояснення визначення властивостей *DateTimePicker* з *Designer.cs* перенесено до конструктора класу *Form1*, тобто. у файл клас *Form1.cs*.

Лістинг 4.48. (examp46r)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp46
{
    public partial class Form1 : Form
    {
      public Form1()
      {
            InitializeComponent();
            this.dateTimePicker1.Location = new System.Drawing.Point(374, 50);
            this.dateTimePicker1.Name = "dateTimePicker1";
            this.dateTimePicker1.Size = new System.Drawing.Size(200, 26);
            this.dateTimePicker1.TabIndex = 0;
            this.dateTimePicker1.ValueChanged +=
            newSystem.EventHandler(this.dateTimePicker1_ValueChanged);
            dateTimePicker1.Format = DateTimePickerFormat.Time;
      }
        private void dateTimePicker1_ValueChanged(object sender, EventArgs e)
        {
            label1.Text = String.Format("Value.ToLongTimeString(): {0}",
            dateTimePicker1.Value.ToLongTimeString());
            label2.Text = String.Format("Value.ToShortDateString(): {0}",
            dateTimePicker1.Value.ToShortDateString());
            label3.Text = String.Format("Text: {0}", dateTimePicker1.Text);
        }
    }
}
```

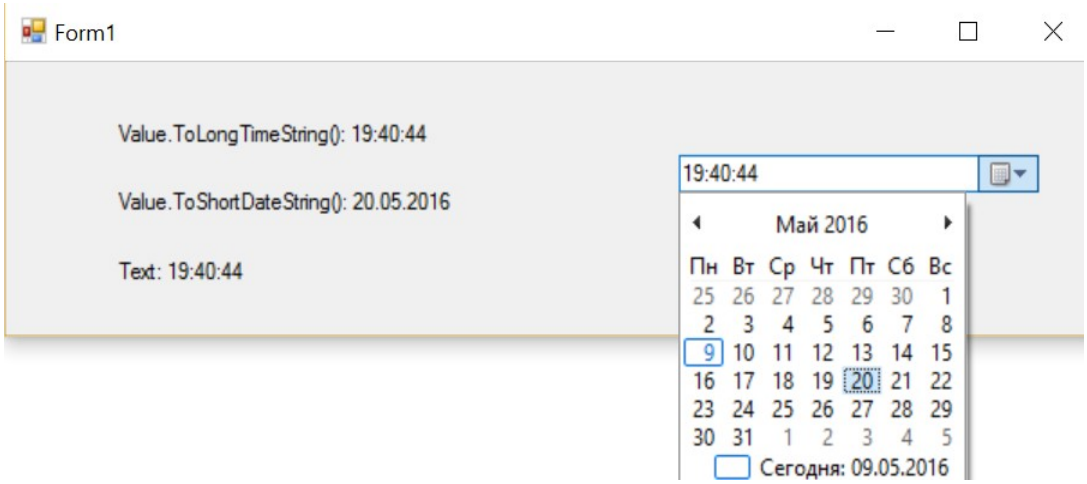


Рис. 4.76. Обрання дати

У наявності є властивості, що дозволяють змінювати зовнішній вигляд календаря, встановлюючи заголовок, кольори тексту і фону для назв місяців. Властивість *ShowUpDown* визначає, чи буде в елементі керування відображатися стрілка *UpDown*. Поточне висвічене значення може змінюватися натисканням по стрілці вгору або вниз. Нижче показано, як створити елемент *DateTimePicker*, що дозволяє користувачу вибирати лише час.

Для цього стосовно попереднього лістингу достатньо в конструкторі *Form1* додати рядок:

```
timePicker.ShowUpDown = true;
```

Результат вирішення наведено на рис. 4.77.

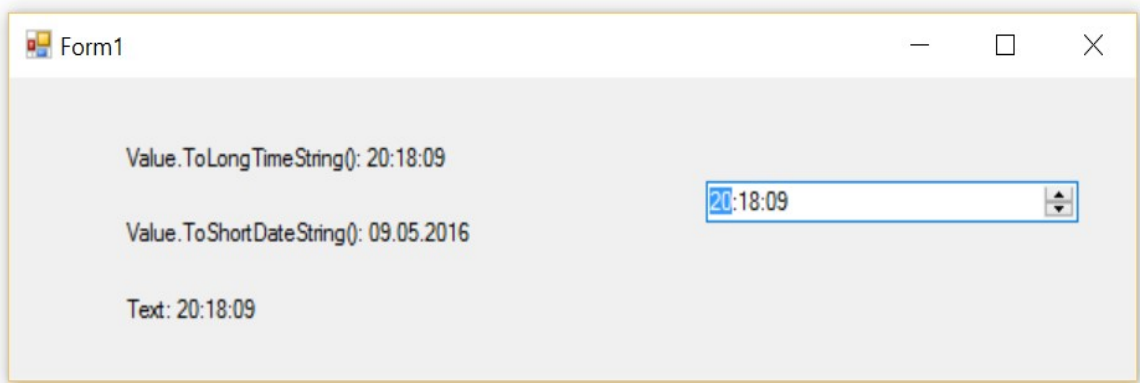


Рис. 4.77. Обрання часу.

MonthCalendar

За допомогою *MonthCalendar* також можна вибрати дату, тільки в цьому випадку цей елемент представляє сам календар, який не потрібно розкривати:

Компонент *MonthCalendar* нагадує компонент *DateTimePicker*, що працює в режимі введення дат. Щоправда, у компоненті *MonthCalendar* передбачено деякі додаткові можливості:

- можна допустити множинний вибір дат у певному діапазоні (властивість *MultiSelect*);
- можна зазначати у календарі номери тижнів з початку року (властивість *WeekNumbers*);

За допомогою *MonthCalendar* також можна вибрати дату, тільки в цьому випадку цей елемент представляє сам календар, який не потрібно розкривати:

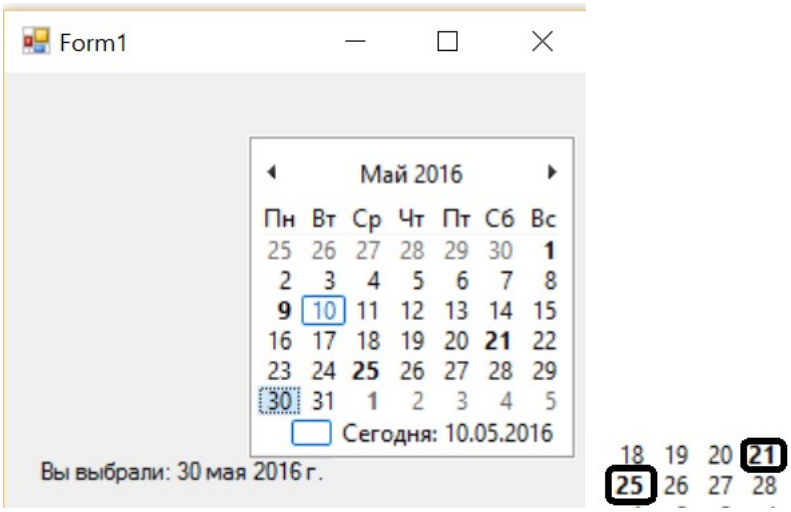


Рис. 4.78. До елементу керування MonthCalendar та виділення днів жирним шрифтом: **1, 9, 21, 25**

Деякі основні властивості елемента.

Властивості виділення дат:

- *AnnuallyBoldedDates*: містить набір дат, які будуть позначені жирним у календарі для кожного року
- *BoldedDates*: містить набір дат, які будуть відзначені жирним (тільки для поточного року)
- *MonthlyBoldedDates*: містить набір дат, які будуть відзначені жирним для кожного місяця

Додавання виділених дат здійснюється за допомогою певних методів:

```
monthCalendar1.AddBoldedDate(new DateTime(2015, 05, 21));
monthCalendar1.AddBoldedDate(new DateTime(2015, 05, 22));
monthCalendar1.AddAnnuallyBoldedDate(new DateTime(2015, 05, 9));
monthCalendar1.AddMonthlyBoldedDate(new DateTime(2015, 05, 1));
```

Для зняття виділення можна використати аналоги цих методів:

```
monthCalendar1.RemoveBoldedDate(new DateTime(2016, 05, 21));
monthCalendar1.RemoveBoldedDate(new DateTime(2016, 05, 22));
monthCalendar1.RemoveAnnuallyBoldedDate(new DateTime(2016, 05, 9));
monthCalendar1.RemoveMonthlyBoldedDate(new DateTime(2016, 05, 1));
```

Найбільш цікавими подіями елемента є події *DateChanged* і *DateSelected*, які виникають під час зміни обраної в елементі дати. Однак слід враховувати, що обрана дата буде представляти першу дату діапазону виділених дат.

Приклад 47.

На рис. 4.78. показаний елемент керування *MonthCalendar* та виділення жирним шрифтом днів: 1, 9, 21, 25. Текс програми наведено у лістингу 4.49.

У дизайнері встановлено елемент керування *MonthCalendar* (ідентифікатор *monthCalendar1*). Для полегшення пояснення програми прийнято властивості *monthCalendar1* за замовчуванням (*Location*, *Drawing*, *Name*, *TabIndex*) перенесено з *Designer.cs* до конструктора класу *Form1*,

тобто. у файл клас *Form1.cs*. Останні чотири властивості в конструкторі класу встановлено програмним чином і призначені для виділення жирним шрифтом чотирьох вищезазначених днів. У програмі використано подію як відгук на виділену дату.

Лістинг 4.49. (examp47)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp47
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            // перенесены из Designer.cs
            //-----------------------------------------------------------
            this.monthCalendar1.Location = new System.Drawing.Point(121, 31);
            this.monthCalendar1.Name = "monthCalendar1";
            this.monthCalendar1.TabIndex = 0;
            // -----------------------------------------------------------
            // Установленные свойства и определение отклика на выделение даты.
            monthCalendar1.AddBoldedDate(new DateTime(2016, 05, 21));
            monthCalendar1.AddBoldedDate(new DateTime(2016, 05, 25));
            monthCalendar1.AddAnnuallyBoldedDate(new DateTime(2016, 05, 9));
            monthCalendar1.AddMonthlyBoldedDate(new DateTime(2016, 05, 1));
            this.monthCalendar1.DateChanged +=
            new System.Windows.Forms.DateRangeEventHandler
            (this.monthCalendar1_DateChanged);
//monthCalendar1.TodayDate= new DateTime(2016, 05, 22);
//monthCalendar1.ShowTodayCircle = true;
//monthCalendar1.ShowToday = false;
//monthCalendar1.SelectionStart = new DateTime(2016, 05, 1);
//monthCalendar1.SelectionEnd = new DateTime(2016, 05, 11);

        }

  private void monthCalendar1_DateChanged(object sender, DateRangeEventArgs e)
        {
        label1.Text = String.Format("Вы выбрали: {0}",
         e.Start.ToLongDateString());
     // или так - аналогичный код
     // label1.Text = String.Format
     // ("Вы выбрали: {0}", monthCalendar1.SelectionStart.ToLongDateString());
      }
  }
```

Властивості визначення дат у календарі:

- *MinDate*: визначає мінімальну дату для вибору календаря
- *MaxDate*: задає максимальну дату для вибору календаря
- *FirstDayOfWeek*: визначає день тижня, з якого вона має починатися

- *SelectionRange*: визначає діапазон виділених дат
- *SelectionEnd*: задає початкову дату виділення
- *SelectionStart*: визначає кінцеву дату виділення
- *ShowToday*: при значенні *true* відображає знизу календаря поточну дату
- *ShowTodayCircle*: при значенні *true* поточна дату буде обведено кружечком
- *TodayDate*: визначає поточну дату. За замовчуванням використовується системна дата на комп'ютері, але за допомогою цієї властивості ми можемо її змінити

Наприклад, при встановленні властивостей (їх виділено у коментарях лістингу 4.49):

```
monthCalendar1.TodayDate= new DateTime(2016, 05, 22);
monthCalendar1.ShowTodayCircle = true;
monthCalendar1.ShowToday = false;
monthCalendar1.SelectionStart = new DateTime(2016, 05, 1);
monthCalendar1.SelectionEnd = new DateTime(2016, 05, 11);
```

буде наступне відображення календаря:

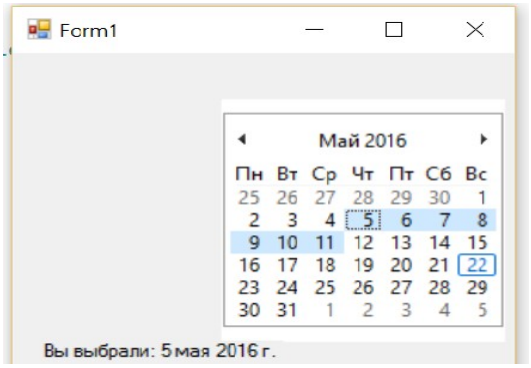


Рис. 4.79. Виділення діапазону чисел

4.22. PictureBox, ProgressBar

PictureBox

Елемент керування *PictureBox* використовується для відображення графічних зображень. Зображення може бути у форматі *BMP*, *JPEG*, *GIF*, *PNG*, метафайлу або піктограми. Властивість *SizeMode* використовує перелік *PictureBoxSizeMode* для визначення того, як зображення розміщується в елементі керування.

- *Normal*: зображення позиціонується у верхньому лівому куті *PictureBox*, і розмір зображення не змінюється. Якщо *PictureBox* більше розмірів зображення, то праворуч і знизу з'являються порожні місця, якщо менше, то зображення обрізається
- *StretchImage*: зображення розтягується або стискається таким чином, щоб умістити по всій ширині та висоті елемента *PictureBox*
- *AutoSize*: *PictureBox* автоматично розтягується, підлаштовуючись під розміри зображення.

- *CenterImage*: якщо *PictureBox* менше зображення, зображення обрізається по краях і виводиться лише його центральна частина. Якщо ж *PictureBox* більше зображення, то воно позиціонується по центру.
- *Zoom*: зображення підлаштовується під розміри *PictureBox*, зберігаючи при цьому пропорції

Розмір зображення *PictureBox* можна змінювати, встановлюючи властивість *ClientSize*. Під час створення *PictureBox* спочатку створюється об'єкт, що базується на *Image*. Наприклад, щоб завантажити файл *JPEG* у *PictureBox*, потрібно зробити так:

```
Bitmap myJpeg = new Bitmap("mypic.jpg");
pictureBox1.Image = (Image)myJpeg;
```

Відзначимо необхідність приведення до типу *Image*, оскільки властивість *Image* елемента керування *PictureBox* має цей тип.

Приклад 48.

Це приклад використання *PictureBox* для відображення графічного растрового зображення у форматі *jpg*. При цьому використовуються три кнопки та *PictureBox*. Перша кнопка забезпечує виведення ліній та еліпсів. Друга кнопка забезпечує очищення *PictureBox*. Третя кнопка виводить растрову картинку, представлену файлом *1.jpg*, що у поточній директорії, тобто у папці *PictureBox.exe*. Текст програми показано у лістингу 4.50. Фрагменти результатів розв'язання задач при натисканні першої та третьої кнопок відповідно показано на рис. 4.80 та 4.81.

Слід зазначити, що значну частину прикладів використання елемента керування *PictureBox* наведено у посібнику "Розробка програмного забезпечення мовою C#, частина 2" [1].

Лістинг 4.50. (examp48)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp48
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click_1(object sender, EventArgs e)
        {
            this.pictureBox1.Size = new System.Drawing.Size(745, 186);
```

```
            pictureBox1.SizeMode = PictureBoxSizeMode.Zoom;
            Graphics g = pictureBox1.CreateGraphics();
            Pen myPen = new Pen(Color.Black);
            Pen myPen2 = new Pen(Color.Red);
            Font drawFont = new Font("Arial", 10);
            SolidBrush br = new SolidBrush(Color.Red);
            Random a = new Random();
            int count = a.Next(2, 5);
            int len = 100 ;/// count;
            Point[] apt = new Point[count + 1];
            Text = count.ToString();
            apt[0] = new Point(20, 71);
            g.FillEllipse(br, 6, 71, 28,28);
            for (int i = 1; i < count + 1; i++)
            {
                apt[i] = new Point(10 + i * len, 71);
                string str = a.Next(10000000, 99999999).ToString();
                g.DrawString(str, drawFont, br, (float)(10 + (i - 1) * len),
                (float)50);
                g.FillEllipse(br, 6 + i * len, 71, 28, 28);
            }
            g.DrawLines(myPen, apt);
            myPen.Dispose();
            myPen2.Dispose();
            br.Dispose();
            g.Dispose();
        }
        private void button2_Click(object sender, EventArgs e)
            {
                    pictureBox1.Image = null;
                    pictureBox1.Invalidate();
            }

        private void button3_Click(object sender, EventArgs e)
        {
            Bitmap image; //Bitmap для открываемого изображения
            OpenFileDialog open_dialog = new OpenFileDialog();
            //создание диалогового окна для выбора файла
            open_dialog.Filter = "ImageFiles(*.BMP;*.JPG;*.GIF;*.PNG)|*.BMP;
          . JPG;*.GIF;*.PNG|All files (*.*)|*.*"; //формат загружаемого файла
            if (open_dialog.ShowDialog() == DialogResult.OK)
            //если в окне была нажата кнопка "OK"
            {
                try
                {
                    image = new Bitmap(open_dialog.FileName);
                    //вместо pictureBox1 укажите pictureBox, в который
                    //нужно загрузить изображение
                    this.pictureBox1.Size = image.Size;
                    pictureBox1.Image = image;
                    pictureBox1.Invalidate();
                }
                catch
                {
                  DialogResult rezult = MessageBox.Show("Невозможно открыть
файл",
                    "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
         }
    }
}
```

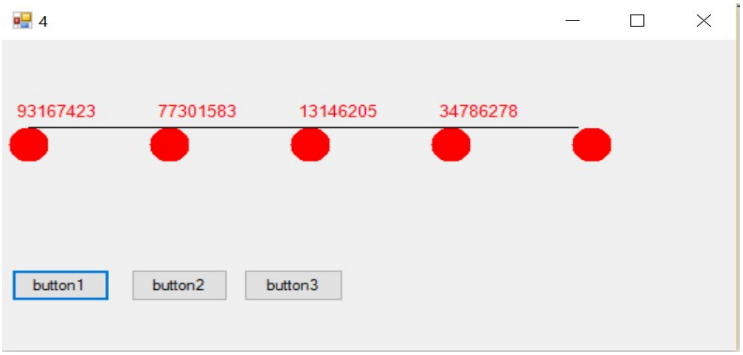


Рис. 4.80. Робота з векторною графікою

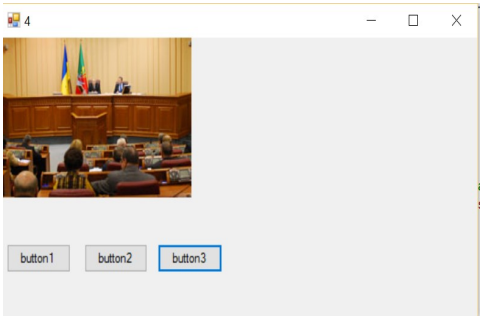


Рис. 4.81. Робота з растровою графікою

ProgressBar

У той час, як елемент керування *TrackBar* призначено для регулювання будь-яких числових значень, елемент керування *ProgressBar* дозволяє графічно відобразити значення. Він часто застосовується, наприклад, для відображення відсотка завершення будь-якого тривалого процесу.

Розглянемо елемент керування *ProgressBar* двох прикладах. У першому прикладі використовуємо елементи керування *TrackBar*, *ProgressBar* та *Label*. При переміщенні повзунка *TrackBar* його положення синхронно графічно відображається в *ProgressBar*. Значення показника, що змінюється під час переміщення повзунка, відображається в *Label*.

На рис. 4.82. наведено фрагмент результату розв’язання задачі.

Рис. 4.82. До використання елемента керування ProgressBar

При додаванні *ProgressBar* у вікно форми, дизайнер форм автоматично створює за допомогою конструктора об’єкт класу *System.Windows.Forms.ProgressBar*:

```
this.progressBar1 = new System.Windows.Forms.ProgressBar();
```

та додає код налаштування властивостей цього елемента керування:

```
this.progressBar1.Location = new System.Drawing.Point(24, 64);
this.progressBar1.Name = "progressBar1";
this.progressBar1.Size = new System.Drawing.Size(224, 23);
this.progressBar1.TabIndex = 2;
```

Розглянемо найважливіші властивості елемента керування *ProgressBar*.

Перш за все програма буде використовувати властивість *Value*, що встановлює поточне значення, пов'язане з елементом керування *ProgressBar*. Саме це значення визначає розмір зафарбованої області вікна *ProgressBar*.

Поточне значення має знаходитись у межах, що задаються за допомогою властивостей *Minimum* та *Maximum*. Перше визначає мінімально можливе значення, що відображається за допомогою *ProgressBar*, а друге - максимально можливе значення. За замовчуванням *ProgressBar* відображає значення від 0 до 100.

При ініціалізації наш додаток встановлює початкову позицію елементів керування *TrackBar* та *ProgressBar*, використовуючи для цього властивість *Value*:

```
trackBar1.Value = 0;
label1.Text = trackBar1.Value.ToString();
progressBar1.Value = trackBar1.Value;
```

Новий обробник повідомлення *trackBar1_Scroll* відображає чисельне значення показника в полі *label1*, а також встановлює нову позицію елемента керування *progressBar1*:

```
private void trackBar1_Scroll(object sender, System.EventArgs e)
{

     // установленная позиция ползунка соответствует
     // значению показателю равному 75, как это показано на рис.
     label1.Text = trackBar1.Value.ToString();
     progressBar1.Value = trackBar1.Value;
}
```

За допомогою методів *Increment* та *PeformStep* програма може збільшувати поточну позицію елемента керування *ProgressBar*. Ці методи зручно використовувати у циклі. Метод *Increment* збільшує позицію на величину, що передається цьому методу через єдиний параметр. Що ж до методу *PeformStep*, то він збільшує значення на крок, заданий за допомогою властивості *Step*.

При необхідності заблокувати *ProgressBar* слід привласнити властивості *Enabled* значення *false*.

Приклад 49.

Приклад полягає у програмуванні формування букета та оцінці його вартості. Як показано на рис. 4.83. формування букета виконується натисканням *RadioButton*. Позначки в *CheckBox* вказують, які квіти входять

до складу вибраного пакета. Програма наведена у лістингу 4.51. *Minimum* і *Maximum* прийнято за замовчуванням, тобто їх значення відповідно дорівнюють 0 і 100. У деяких пізніх версіях *C#* ці властивості використовуються лише за замовчуванням (властивості мають лише параметр *get{}*). У нашому прикладі *progressBar1.Value(X)* в залежності від реального максимального значення X_{max} і вартості букета $X_{бук}$ буде:

$$X = (int)\ (100.0 / X_{max} * X_{бук});$$

У прикладі максимальна вартість букета X_{max}= *120 грн*.

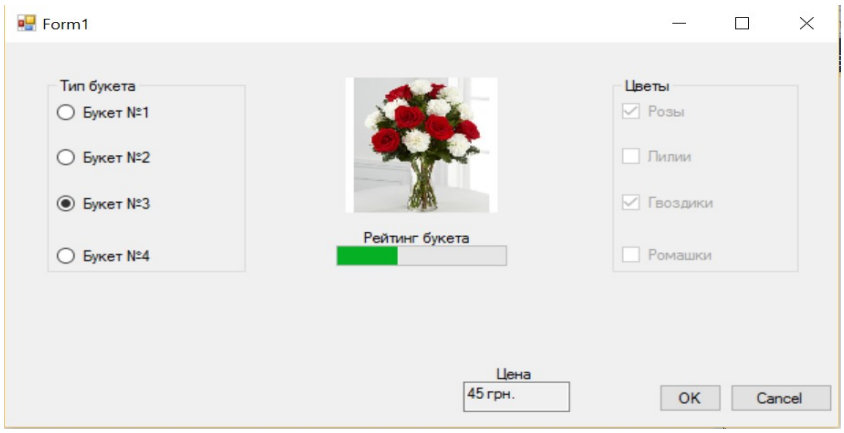


Рис. 4.83. Формування букету квітів

Лістинг 4.51. (examp49)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp49
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            radioButton1.Checked = true;
            textBox1.ReadOnly = true;
        }

        private void radioButton1_CheckedChanged(object sender, EventArgs e)
        {
            checkBox1.Checked = true;
            checkBox2.Checked = true;
            checkBox3.Checked = true;
            checkBox4.Checked = true;
            textBox1.Text = "120 грн.";
            progressBar1.Value = progressBar1.Maximum;
            pictureBox1.Image = Image.FromFile("toys\\1.jpg");
        }
```

```
        private void radioButton2_CheckedChanged(object sender, EventArgs e)
        {
            checkBox1.Checked = true;
            checkBox2.Checked = true;
            checkBox3.Checked = false;
            checkBox4.Checked = true;
            textBox1.Clear();
            textBox1.Text = "90 грн.";
            progressBar1.Value = (int)(100.0 / 120 * 90);
            pictureBox1.Image = Image.FromFile("toys\\2.jpg");
        }

        private void radioButton3_CheckedChanged(object sender, EventArgs e)
        {
            checkBox1.Checked = true;
            checkBox2.Checked = false;
            checkBox3.Checked = true;
            checkBox4.Checked = false;
            textBox1.Text = "45 грн.";
            progressBar1.Value = (int)(100.0 / 120 * 45);
            pictureBox1.Image = Image.FromFile("toys\\3.jpg");
        }

        private void radioButton4_CheckedChanged(object sender, EventArgs e)
        {
            checkBox1.Checked = true;
            checkBox2.Checked = false;
            checkBox3.Checked = false;
            checkBox4.Checked = false;
            textBox1.Text = "30 грн.";
            progressBar1.Value = (int)(100.0 / 120 * 30);
            pictureBox1.Image = Image.FromFile("toys\\4.jpg");
        }

        private void button1_Click(object sender, EventArgs e)
    {
    if (radioButton1.Checked == true)
     MessageBox.Show("Вы приобрели букет за " + textBox1.Text, "Ваша покупка");
    if (radioButton2.Checked == true)
     MessageBox.Show("Вы приобрели букет за " + textBox1.Text, "Ваша покупка");
    if (radioButton3.Checked == true)
    MessageBox.Show("Вы приобрели букет за " + textBox1.Text, "Ваша покупка");
    if (radioButton4.Checked == true)
    MessageBox.Show("Вы приобрели букет за " + textBox1.Text, "Ваша покупка");
    this.Close();
    }

        private void button2_Click(object sender, EventArgs e)
        {
          const string message =
          "Вы не совершили покупку. Желаете продолжить выход?";
          const string caption = "Выход";
          var result = MessageBox.Show(message, caption,
          MessageBoxButtons.YesNo,MessageBoxIcon.Question);
          if (result == DialogResult.Yes)
          this.Close();
        }
    }
}
```

4.23. Елемент керування DataGridView

Елемент керування *DataGridView* надає потужний та гнучкий спосіб відображення даних у табличному форматі. Цей елемент можна використовувати для представлення в режимі читання невеликих об'єктів даних; можна розширити цей елемент для представлення великих обсягів даних як редагування. Функціональні можливості елемента керування *DataGridView* можна розширити кількома способами, щоб реалізувати поведінку користувача в додатках. Наприклад, можна програмно задати власні алгоритми сортування, і навіть створити власні типи осередків. Зовнішній вигляд елемента керування *DataGridView* можна налаштувати, надавши кілька властивостей. Як джерело даних можна використовувати різні типи сховищ даних. Крім того, елемент керування *DataGridView* може працювати без зв'язаних джерел даних.

DataGridView підтримує стандартну модель прив'язки даних *Windows Form*, яка допускає прив'язку до екземплярів класів, опис яких наведено у наступному списку.

- Будь-який клас, що реалізовує інтерфейс *IList*, включаючи одновимірні масиви.
- Будь-який клас, що реалізує інтерфейс *IListSource*, наприклад, класи *DataTable* і *DataSet*.
- Будь-який клас, що реалізує інтерфейс *IBindingList*, наприклад, клас *BindingList<T>*.
- Будь-який клас, що реалізує інтерфейс *IBindingListView*, наприклад клас *BindingSource*.

Зазвичай прив'язка здійснюється до компоненту *BindingSource*, а компонент *BindingSource* зв'язується з іншим джерелом даних або заповнюється бізнес-об'єктами. Компонент *BindingSource* кращий, оскільки забезпечує прив'язку до різних типів джерел даних і може автоматично вирішувати різні проблеми прив'язування даних. Додаткові відомості наведено нижче під час роботи з базами даних з використанням технології *ADO NET*.

Елемент керування *DataGridView* також може застосовуватися в незв'язаному режимі без базового сховища даних. Елемент керування *DataGridView* можна легко налаштувати та розширити. Крім того, елемент керування надає безліч властивостей, методів та подій, що дозволяють налаштувати зовнішній вигляд та поведінку. Якщо ви бажаєте відображати табличні дані за допомогою *Windows Forms*, розгляньте можливість використання елемента керування *DataGridView*, перш ніж розглядати інші рішення (наприклад, *DataGrid*). Якщо необхідно виконати відображення сітки з невеликим об'ємом даних лише для читання або надання користувачу можливості редагування таблиці з мільйонами записів, можна використовувати елемент керування *DataGridView*, який відрізняється ефективним використанням пам'яті та швидким програмуванням.

Коротко розглянемо основні моменти роботи *DataGridView*.

1. Спочатку встановлюємо форму компонент *DataGridView*. Джерело даних при створенні можна не вказувати, оскільки *DataGridView* дозволяє зберігати дані в собі і додавати/видаляти під час виконання (*runtime*). Після цього редактором властивостей рекомендуємо встановити властивості *AllowUserToAddRows* та *AllowUserToDeleteRows* у *false*, *ReadOnly* у *true*. Можна також перейменувати екземпляр з *dataGridView1* на щось більш осмислене, що підходить для програми.

2. Стовпці та рядки можна додавати під час виконання програми. Однак якщо призначення та структура таблиці (кількість та найменування стовпців) відомі заздалегідь, можна редактором властивостей додати стовпці. Для цього редагують властивість *Columns*.

3. Звернення до осередків *DataGridView* досить просте. Індексація стовпців (*Columns*) та рядків (*Rows*) йде до нуля. Стовпець з індексом 0 найлівіший, і рядок з індексом 0 найвищий. Кількість рядків можна отримати через властивість *RowCount*. Додаються рядки методом *Rows.Add()*, а видаляються методом *Rows.RemoveAt(номер рядка)*. Приклад додавання та видалення рядків:

```
//добавим в dataGridView1 4 строки
                                dataGridView.Rows.Add();
                                dataGridView1.Rows.Add();
                                dataGridView1.Rows.Add();
                                dataGridView1.Rows.Add();

                                //удалим все строки из dataGridView1
                                while (0 != dataGridView1.RowCount)
                                dataGridView1.Rows.RemoveAt(0);
```

4. Дізнатися номер поточного рядка (i) та стовпця (j) можна за допомогою *CurrentCellAddress*:

```
int i = dataGridView1.CurrentCellAddress.Y;
int j = dataGridView1.CurrentCellAddress.X;
```

Встановити (змінити) поточний рядок можна за допомогою властивості *CurrentCell* (у прикладі встановлюємо третій поточний рядок):

```
int idx = 2;
dataGridView1.CurrentCell = dataGridView1.Rows[idx].Cells[0];
```

Поточний рядок позначений у таблиці трикутником:

5. Змінювати дані в комірках таблиці можна лише тоді, коли існують відповідний стовпець і відповідний рядок, інакше буде помилка. (Індекс повинен бути позитивним числом, а його розмір не повинен перевищувати розмір колекції.). Після додавання рядка методом *Add* усі значення в комірках доданого рядка будуть порожні. До комірки таблиці можна звертатися за номером стовпця та номером рядка через властивість *Value*. Приклад додавання та заповнення таблиці:

```
for (int i = 0; i < dataGridView1.RowCount; i++)
{
   dataGridView1.Rows.Add();
   dataGridView1.Rows[i].Cells[0].Value = (i+1).ToString();
   dataGridView1.Rows[i].Cells[1].Value = "пример текста";
}
```

Розглянемо приклади формування та роботи з таблицями з використанням елемента керування *DataGridView*. У всіх наведених прикладах цей елемент керування переноситься з *ToolBox* у форму з ім’ям *DataGridView1*.

Приклад 50.

Тут у режимі редактора форми додатково встановлюються дві кнопки. При натисканні першої кнопки виводиться перша таблиця із двома колонками, при натисканні другої кнопки виводиться друга таблиця з чотирма колонками. Це можна побачити на рис. 4.84. Відмінна риса формування таблиць досить наочно показано у лістингу програми 4.52. Як бачимо, у програмі використовується об’єкт-таблиця даних *DataTable*. З його допомогою спочатку заповнюємо "шапку" таблиці даних, використовуючи метод *Coloums.Add*, а потім безпосередньо комірки таблиці, використовуючи метод *Rows.Add*. Щоб передати побудовану таблицю елементу керування *DataGridView*, вказуємо як джерело даних *DataSource* об’єкт-таблиця класу *DataTable*. Натискаючи мишею по заголовках колонок, отримуємо сортування даних в алфавітному порядку.

При натисканні на другу кнопку пропонується ще один варіант формування таблиці. При цьому кожній колонці визначається тип даного. Це важливо під час контролю введення даних.

Лістинг 4.52. (examp50)

```
using System;
using System.Data;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
программе

namespace examp50
{
    public partial class Form1 : Form
    {
        DataTable table;
        public Form1()
        {
```

```
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            table = new DataTable(); // New data table.
            base.Text = "Формирование таблицы";
            String[] Imena = {"Андрей - раб", "Света-Х", "ЖЭК",
                      "Справка по тел", "Александр Степанович", "Мама - дом",
                      "Карапузова Таня", "Погода сегодня", "Театр Браво"};
            String[] Tel = {"274-88-17", "+38(067)7030356",
                      "22-345-72", "009", "223-67-67 доп 32-67", "570-38-76",
                      "201-72-23-прямой моб", "001", "216-40-22"};
            // Создание объекта table данных
            // DataTable table = new DataTable();
            // Заполнение шапки таблицы
            table.Columns.Add("Имена");
            table.Columns.Add("Номера телефонов ");
            // Заполнение клеток (ячеек) таблицы данных
            for (int i = 0; i <= 8; i++)
            table.Rows.Add(new String[] { Imena[i], Tel[i] });
            // Для сетки данных указываем источник данных
            dataGridView1.DataSource = table;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            table = new DataTable(); // New data table.
            table.Columns.Add("Dosage", typeof(int)); // Add five columns.
            table.Columns.Add("Drug", typeof(string));
            table.Columns.Add("Patient", typeof(string));
            table.Columns.Add("Date", typeof(DateTime));
      table.Rows.Add(15, "Abilify", "Jacob", DateTime.Now); // Add five data
rows.
            table.Rows.Add(40, "Accupril", "Emma", DateTime.Now);
            table.Rows.Add(40, "Accutane", "Michael", DateTime.Now);
            table.Rows.Add(20, "Aciphex", "Ethan", DateTime.Now);
            table.Rows.Add(45, "Actos", "Emily", DateTime.Now);
            dataGridView1.DataSource = table;
        }
    }
}
```

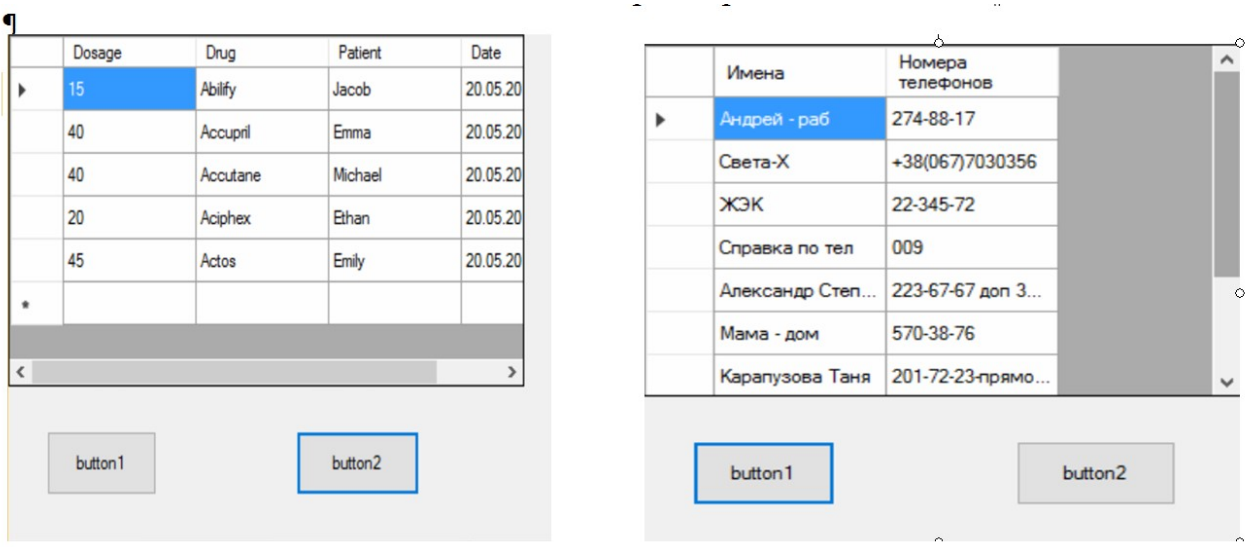


Рис. 4.84. Форматування таблиць

Приклад 51.

У цьому прикладі показано розширені можливості створення таблиці. Досить прокоментований текст програми, наведено у лістингу 4.53. Тому додаткові описи не потрібні. На рис. 4.85. наведено фрагмент виконання програми.

Лістинг 4.53. (examp51)

```
using System;
using System.Data;
using System.Windows.Forms;

namespace examp51
{
    public partial class Form1 : Form
    {
        DataTable table;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            //создадим таблицу вывода товаров с колонками
            //Название, Цена, Остаток
            var column1 = new DataGridViewColumn();
            column1.HeaderText = "Название"; //текст в шапке
            column1.Width = 100; //ширина колонки
            column1.ReadOnly = true; //значение в этой колонке нельзя править
      //текстовое имя колонки, его можно использовать вместо обращений по
индексу
            column1.Name = "name";
            //флаг, что данная колонка всегда отображается на своем месте
            column1.Frozen = true;
        column1.CellTemplate = new DataGridViewTextBoxCell(); //тип нашей
колонки
            var column2 = new DataGridViewColumn();
            column2.HeaderText = "Цена";
            column2.Name = "price";
            column2.CellTemplate = new DataGridViewTextBoxCell();
            var column3 = new DataGridViewColumn();
            column3.HeaderText = "Остаток";
            column3.Name = "count";
            column3.CellTemplate = new DataGridViewTextBoxCell();
            dataGridView1.Columns.Add(column1);
            dataGridView1.Columns.Add(column2);
            dataGridView1.Columns.Add(column3);
            //запрешаем пользователю самому добавлять строки
            dataGridView1.AllowUserToAddRows = false;
            for (int i = 0; i < 5; ++i)
            {
            //Добавляем строку, указывая значения колонок поочереди слева
направо
            dataGridView1.Rows.Add("Пример 1, Товар " + i, i * 1000, i);
            }
            for (int i = 0; i < 5; ++i)
            {
            //Добавляем строку, указывая значения каждой ячейки по имени
            //(можно использовать индекс 0, 1, 2 вместо имен)
```

```
                dataGridView1.Rows.Add();
                dataGridView1["name",dataGridView1.Rows.Count - 1].Value =
                "Пример 2, Товар " + i;
                dataGridView1["price", dataGridView1.Rows.Count - 1].Value = i *
1000;
                dataGridView1["count", dataGridView1.Rows.Count - 1].Value = i;
                }
                //А теперь простой пройдемся циклом по всем ячейкам
                for (int i = 0; i < dataGridView1.Rows.Count; ++i)
                {
                    for (int j = 0; j < dataGridView1.Columns.Count; ++j)
                    {
                    //Значения ячеек хряняться в типе object
                    //это позволяет хранить любые данные в таблице
                    object o = dataGridView1[j, i].Value;
                    }
                }
            }
        }
    }
```

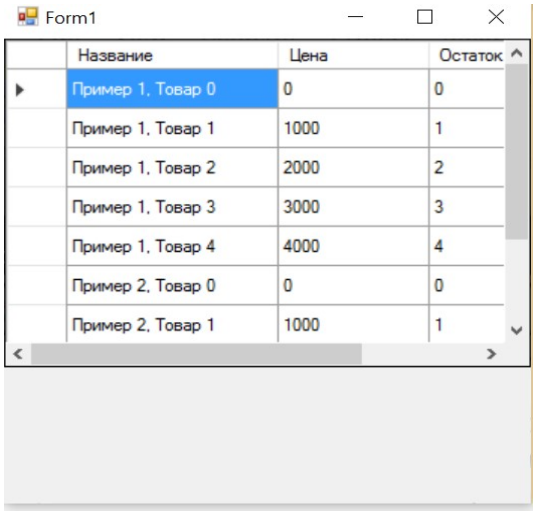


Рис. 4.85. Виведення товарів

Приклад 52.

У програмі використовується клас *BindingList<T>*, який забезпечує прив'язку даних. Використання такого класу дозволяє при оновленні (додаванні) даних в об'єкті класу *BindingList* забезпечити автооновлення об'єкта класу *DataGridView*. Текст програми та результат її виконання відповідно наведено у лістингу 4.54 та рис. 4.86. У програмі об'єкт *data*, що є джерелом даних, містить перелік об'єктів класу *SampleRow*.

Лістинг 4.54. (examp52)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
namespace examp52
{
    class SampleRow
    {
        public string Name { get; set; } //обязательно нужно использовать get
конструкцию
            public float Price { get; set; }
            public int Count { get; set; }
        public string Hidden = ""; //Данное свойство не будет отображаться как
колонка
            public SampleRow(string name, float price, int count)
            {
                this.Name = name;
                this.Price = price;
                this.Count = count;
            }
        }
        public partial class Form1 : Form
            {
                public Form1()             {
                    InitializeComponent();
                }
            private void Form1_Load(object sender, EventArgs e)
            {
                  BindingList<SampleRow> data = new BindingList<SampleRow>();
                  data.Add(new SampleRow("Товар 1", 100, 1));
                  data.Add(new SampleRow("Товар 2", 200, 2));
                  data.Add(new SampleRow("Товар 3", 300, 3));
                  dataGridView1.DataSource = data;
                  data.Add(new SampleRow("Товар 4", 400, 4));
            }
        }
}
```

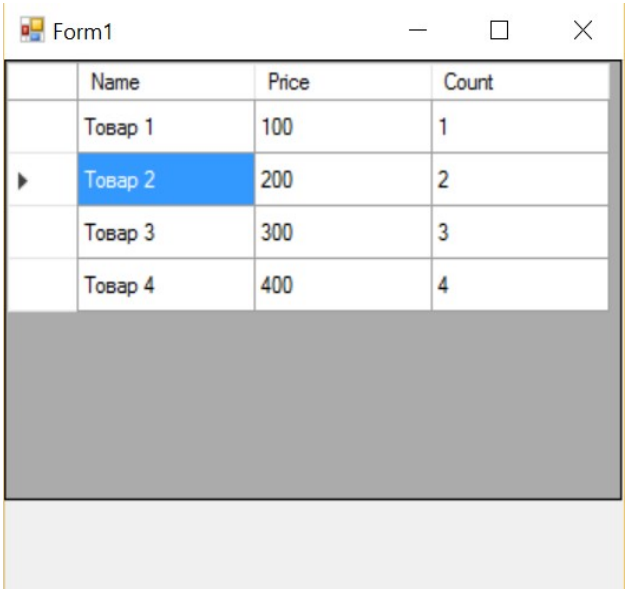


Рис. 4.86. До використання класу BindingList<T>

Приклад 53.

У цьому прикладі виконується виведення таблиці та контроль введення цілих чисел у першу колонку. Ця колонка описана для введення цілих цілих чисел (типу int). Програму наведено у лістингу 4.55. Фрагмент результату роботи програми наведено на рис. 4.87.

Лістинг 4.55. (examp53)

```
using System;
using System.Data;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной
программе

namespace examp53
{
    public partial class Form1 : Form
    {
        DataTable table;
        public Form1()
        {
            InitializeComponent();
            table = new DataTable(); // New data table.
            table.Columns.Add("Dosage", typeof(int)); // Add five columns.
            table.Columns.Add("Drug", typeof(string));
            table.Columns.Add("Patient", typeof(string));
            table.Columns.Add("Date", typeof(DateTime));
            // Add five data rows.
            table.Rows.Add(15, "Abilify", "Jacob", DateTime.Now);
            table.Rows.Add(40, "Accupril", "Emma", DateTime.Now);
            table.Rows.Add(40, "Accutane", "Michael", DateTime.Now);
            table.Rows.Add(20, "Aciphex", "Ethan", DateTime.Now);
            table.Rows.Add(45, "Actos", "Emily", DateTime.Now);
            dataGridView1.DataSource = table;
        }

        //Проверка ячейки
        private void dataGridView1_CellValidating(object sender,
         DataGridViewCellValidatingEventArgs e)
        {
             try
            {
                if (dataGridView1.Columns[e.ColumnIndex].Name == "Dosage")
                {
                    if (Convert.ToInt32(e.FormattedValue) < 1
                        || Convert.ToInt32(e.FormattedValue) > 200)
                    {
                        dataGridView1.Rows[e.RowIndex].ErrorText =
                    "  Значение поля код должно лежать в диапазоне от 1 до 200";
                        e.Cancel = true;
                    }
                    else dataGridView1.Rows[e.RowIndex].ErrorText = null;
                }
            }

             catch (Exception ex)
            {
                dataGridView1.Rows[e.RowIndex].ErrorText = ex.Message;
                e.Cancel = true;
            }
        }

        private void dataGridView1_EditingControlShowing(object sender,
        DataGridViewEditingControlShowingEventArgs e)
        {
            e.Control.KeyPress += new KeyPressEventHandler(Cell_KeyPress);
        }

        private void Cell_KeyPress(object Sender, KeyPressEventArgs pressE)
```

```
        {
            if (!Char.IsDigit(pressE.KeyChar) && pressE.KeyChar != 8)
            pressE.Handled = true;
        }

        private void Delete_Click(object sender, EventArgs e)
        {
            if (dataGridView1.CurrentRow != null)
            dataGridView1.Rows.Remove(dataGridView1.CurrentRow);
        }
    }
}
```

У програмі виконується контроль під час введення кожного символу. Цим символом може бути цифра або код клавіші *BackSpace*. Відгук при натисканні клавіші у полі комірки таблиці виконується у функції:

```
private void dataGridView1_EditingControlShowing(object sender,
DataGridViewEditingControlShowingEventArgs e),
```

в якій згенеровано відгук на натискання кнопки для елемента *Control*.

Якщо при введенні символу, відмінного від цифри та коду клавіші *BackSpace*, він не відображається у поточній комірці.

Крім контролю за введенням кожної введеної цифри, необхідно виконати контроль повністю введеного числа. Цей контроль виконується при зміні фокусування. Наприклад, при переході до іншої комірки. У цьому випадку підключається функція відгуку:

```
private void dataGridView1_CellValidating(object sender,
DataGridView CellValidatingEventArgs e)
```

У цій функції встановлено діапазон 1-200. У разі відхилення від заданого діапазону рядок таблиці виділяється знаком оклику (рис. 4.87). Поки не буде введено допустиме значення, у фокусі буде лише поточна комірка.

Крім контролю даних наведено фрагмент програми видалення позначених рядків у таблиці.

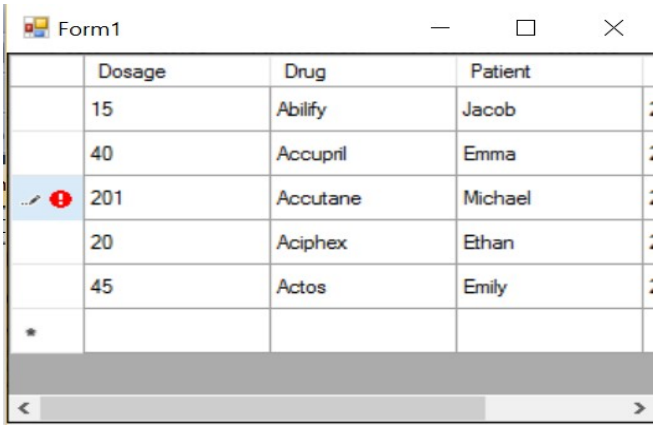


Рис. 4.87. Контроль введення даних

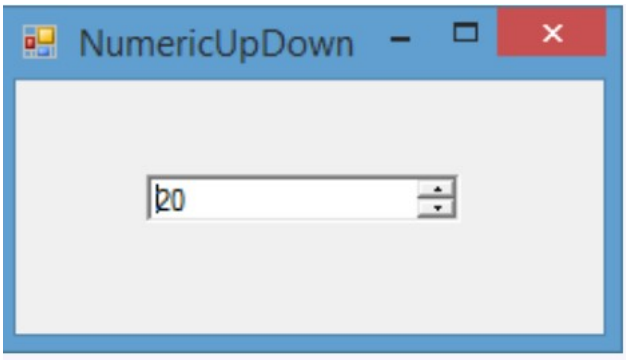
4.24. Елементи NumericUpDown та DomainUpDown

NumericUpDown

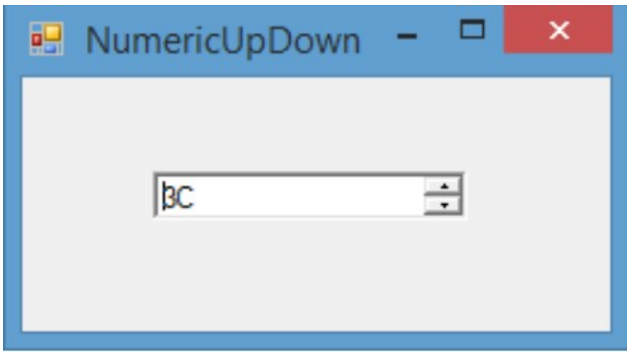
Елемент керування Windows Forms *NumericUpDown* є поєднанням текстового поля і пари кнопок зі стрілками для вибору значення користувачем. Він виводить та задає окреме числове значення у списку варіантів. Користувач може збільшувати та зменшувати число, натискаючи кнопки зі стрілками вгору та вниз або клавіші зі стрілками *ВГОРУ* та *ВНИЗ*, а також вводячи число в поле. При натисканні клавіші зі стрілкою *ВГОРУ* значення збільшується до максимуму; при натисканні клавіші зі стрілкою *ВНИЗ* число зменшується до мінімуму. Одним із варіантів застосування цього елемента керування є його використання як регулятора гучності музичного програвача. Числові елементи керування з можливістю переміщення вгору та вниз за шкалою використовуються в деяких програмах панелі керування *Windows*.

Для визначення діапазону чисел для вибору *NumericUpDown* має дві властивості: *Minimum* (задає мінімальне число) та *Maximum* (задає максимальне число).

Саме значення елемента зберігається як *Value*:



За замовчуванням елемент відображає десяткові числа. Однак якщо ми встановимо його властивість *Hexadecimal* рівною *true*, то елемент відображатиме всі числа у шістнадцятковій системі.



Навіть якщо ми в коді встановимо звичайне десяткове значення:

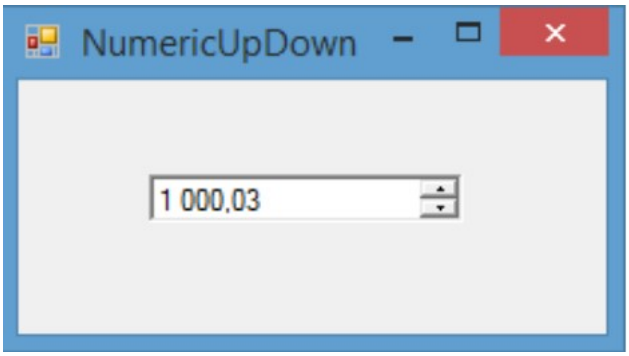
```
numericUpDown1.Value = 66;
```

то елемент все одно відобразить його в шістнадцятковій системі.

Якщо ми хочемо відображати в полі дробові числа, можна використовувати властивість *DecimalPlaces*, яка вказує, скільки знаків після коми має відображатися. За замовчуванням ця властивість дорівнює нулю.

Також можна задати відображення тисячного роздільника. Для цього для властивості *ThousandsSeparator* необхідно встановити значення *true*. Наприклад, *numericUpDown* при

`Value=1000,03, DecimalPlaces=2 и ThousandsSeparator=true:`



При цьому треба враховувати, що якщо ми встановлюємо значення для властивості *Value* у вікні властивостей, то як роздільник цілої і дробової частини використовується кома. Якщо ж ми встановлюємо цю властивість у коді, тоді як роздільником є крапка.

За замовчуванням при натисканні по стрілочкам вгору-вниз, на елементі значення збільшуватиметься або зменшуватиметься на одиницю. Але за допомогою властивості *Increment* можна задати інший крок збільшення, у тому числі й дробовий.

Працюючи з *NumericUpDown* слід враховувати, що його властивість *Value* (як і властивості *Minimum* і *Maximum*) зберігає значення *decimal*. Тому в коді ми також повинні з ним працювати як з *decimal*, а не як з типом *int* або *double*.

Приклад 54.

На лістингу 4.56. та рис. 4.88. наводяться відповідно програма та результат виконання роботи з *NumericUpDown*.

Лістинг 4.56 (examp54)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp54
{
    public partial class Form1 : Form
    {
```

```
        System.Windows.Forms.NumericUpDown numericUpDown1;
        public Form1()
        {
            numericUpDown1 = new System.Windows.Forms.NumericUpDown();
            InitializeComponent();
            numericUpDown1.Parent = this;
            numericUpDown1.Location = new Point(76, 26);
            numericUpDown1.Size = new Size(66, 18);
            numericUpDown1.TextAlign = HorizontalAlignment.Left;
            numericUpDown1.ValueChanged +=
            new EventHandler(numericUpDown1_ValueChanged);
            numericUpDown1.DecimalPlaces = 2;
            numericUpDown1.Increment = 0.1m;
            numericUpDown1.Minimum = -5.00m;
            numericUpDown1.Maximum =  5.00m;
            numericUpDown1.Value = 0.1m;
        }
        private void numericUpDown1_ValueChanged(object sender, EventArgs e)
        {
         label1.Text = numericUpDown1.Value.ToString();
        }
    }
}
```

Під час розробки програми, дизайнером у форму вміщено лише елемент керування *Label*. Для повнішого вивчення *NumericUpDown* робота з ним виконується повністю в програмному коді.

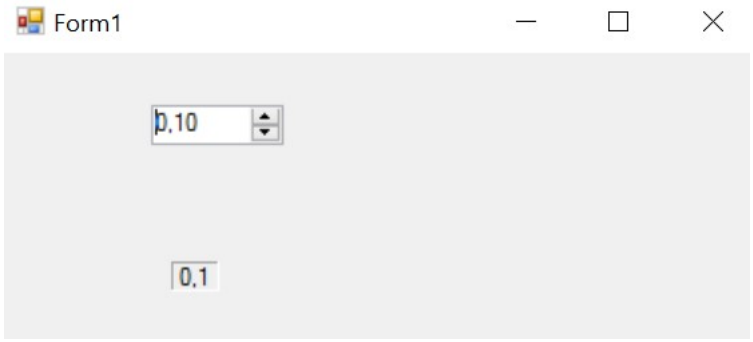


Рис. 4.88. Робота з *NumericUpDown*

DomainUpDown

Елемент керування Windows Forms *DomainUpDown* виглядає як поєднання текстового поля та пари кнопок для переміщення вгору та вниз за списком. Він виводить та задає текстовий рядок у списку варіантів. Користувач може вибрати рядок, переміщаючись за списком за допомогою кнопок зі стрілками вгору і вниз, за допомогою клавіш зі стрілками *ВГОРУ* і *ВНИЗ* або ввівши рядок, що збігається з елементом у списку. Один з можливих способів застосування цього елемента керування - вибір елементів зі списку імен, розташованих в алфавітному порядку. (Для сортування списку властивості *Sorted* необхідно привласнити значення *true*.) За своїм призначенням даний елемент керування близький до списку або поля зі списком, але він більш компактний.

Ключовими властивостями елемента керування є *Items*, *ReadOnly* та *Wrap*. Властивість *Items* містить список об’єктів, текстові значення яких

відображаються в елементі керування. Якщо для властивості *ReadOnly* встановлено значення *false*, елемент керування автоматично завершує текст, який вводиться користувачем, і зіставляє його зі значенням у списку. Якщо для властивості *Wrap* встановлено значення *true*, при прокручуванні до останнього елемента слідом за ним виводиться перший елемент списку і навпаки. Основними методами цього елемента керування є методи *UpButton* та *DownButton*.

Приклад 55.

Код програми використання *DomainUpDown* та результат її виконання наведено відповідно до лістингу 4.57. та рис. 4.89.

Лістинг 4.57. (examp55)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp55
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            List<string> states = new List<string>
             {
               "Аргентина", "Бразилия", "Венесуэла", "Колумбия", "Чили"
             };

            // добавляем список элементов
           // domainUpDown1.Items.AddRange(states);
            domainUpDown1.Items.AddRange(states);
            domainUpDown1.TextChanged += domainUpDown1_TextChanged;
        }

        private void domainUpDown1_TextChanged(object sender, EventArgs e)
        {
            MessageBox.Show(domainUpDown1.Text);
        }
    }
}
```

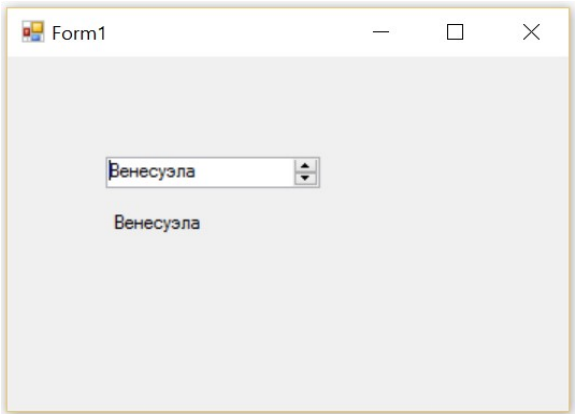


Рис. 4.89. Робота з DomainUpDown

4.25. Клас Environment

Приклад 56.

Клас *Environment* містить низку статичних членів, що дозволяють отримати інформацію щодо операційної системи, в якій виконується *.NET-додаток*. У лістингу 4.58 та на рис. 4.90 відповідно наведено програму та результат її виконання роботи з класом *Environment*. Наведені коментарі у програмі не потребують додаткових пояснень.

Лістинг 4.58. (examp56)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp56 {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics gr = e.Graphics;
            int y = 0;
            string str;
            Brush br = new SolidBrush(ForeColor);
            string[] drives = Environment.GetLogicalDrives();
            int prir = y += 2 * Font.Height;
            for (int i = 0; i < drives.Length; i++)
                str = "Диск : " + i + " - " + drives[i];

                gr.DrawString(str, Font, br, 0, y);
                y += prir/2;
            }
                 y += prir;
                 // Информация об операционной системе.
                 gr.DrawString("Операционная система: " +
Environment.OSVersion,
                 Font, br, 0, y);
```

```
                    y += prir;
                    // Каталог, в котором находится приложение.
                    gr.DrawString("Каталог: " + (Environment.CurrentDirectory),
                    Font, br, 0, y);
                    y += prir;
                    // Версия .NET-платформы, выполняемая на машине.
                    gr.DrawString("Версия .NET-платформы: " + Environment.Version,
                    Font, br, 0, y);
                    y += prir;
                    // Системная папка
                    gr.DrawString("Системная папка: " +
Environment.SystemDirectory,
                    Font, br, 0, y);
                    y += prir;
                    // Имя текущей машины
                    gr.DrawString("Имя текущей машины: " +
Environment.MachineName,
                    Font, br, 0, y);
                }
            }
        }
```

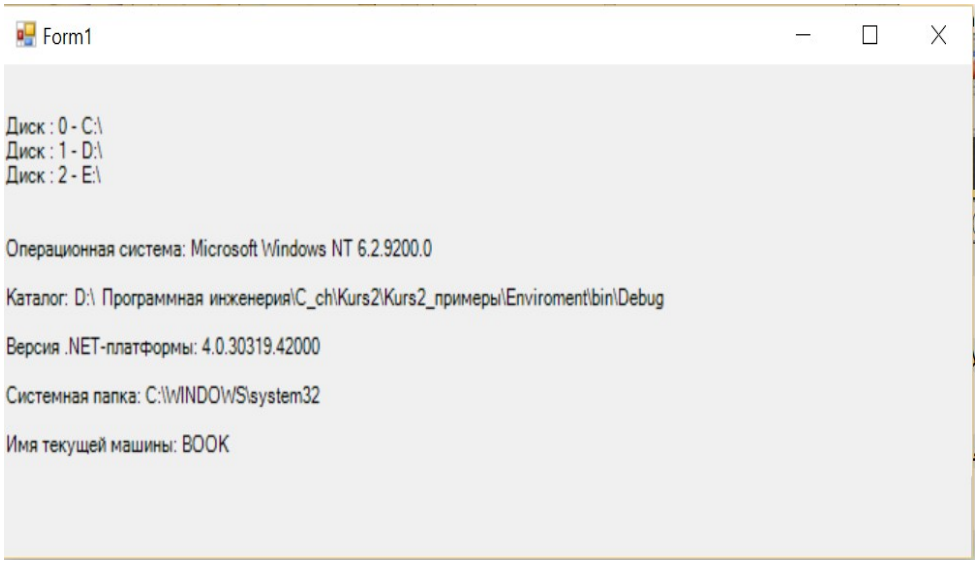


Рис. 4.90. До використання класу Environment

4.26. Елемент керування WebBrowser

Елемент керування *WebBrowser* надає керовану оболонку елемента керування *ActiveX WebBrowser*. Керована оболонка дозволяє відображати веб-сторінки в клієнтських програмах *Windows Forms*. За допомогою *WebBrowser* можна відтворити у програмі функціональність веб-браузера *Internet Explorer*, або можна вимкнути функціональність *Internet Explorer*, задану за замовчуванням, і використовувати цей елемент керування як простий засіб перегляду *HTML-документів*. Цей елемент керування може також використовуватися для додавання на форму *DHTML-елементів* інтерфейсу користувача, причому факт їх розміщення в елементі керування *WebBrowser* буде непомітний. Такий підхід дозволяє ефективно поєднувати використання веб-елементів керування та елементів керування *Windows Forms* у межах однієї програми.

Елемент керування *WebBrowser* має ряд властивостей, методів та подій, які можуть використовуватися для реалізації елементів керування *Internet Explorer*. Наприклад, метод *Navigate* можна використовувати для реалізації адресного рядка, а методи *GoBack*, *GoForward*, *Stop* та *Refresh* – для реалізації кнопок переходу в панелі інструментів. Шляхом обробки події *Navigated* може проводитись оновлення адресного рядка значенням властивості *Url*, а рядки заголовка - значенням властивості *DocumentTitle*.

Створити вміст сторінки в програмі можна шляхом встановлення значення властивості *DocumentText*. За наявності досвіду роботи з об'єктною моделлю *HTML-документів* (*DOM*) вмістом поточної *веб-сторінки* можна керувати за допомогою властивості *Document*. Завдяки цій властивості можна зберігати та змінювати документи в пам'яті замість операцій із файлами.

Властивість *Document* також дозволяє звертатися до методів, реалізованих у коді скрипту *веб-сторінки*, із коду клієнтської програми. Для доступу до коду клієнтської програми з коду скрипта слід задати значення властивості *ObjectForScripting*. Доступ до зазначеного об'єкта з коду скрипта може здійснюватися як до об'єкта *window.external*.

Таблиця 4.7

Властивості та методи *WebBrowser*

Им'я	**Призначення**	**Опис**
Document	властивість	Повертає об'єкт, який забезпечує керований доступ до об'єктної моделі HTML-документа (DOM) для поточної веб-сторінки.
DocumentCompleted	подія	Настає після завантаження веб-сторінки.
DocumentText	властивість	Отримує або задає HTML поточної веб-сторінки
DocumentTitle	властивість	Повертає заголовок поточної веб-сторінки
GoBack	метод	Перехід до попередньої сторінки згідно з хронологією
GoForward	метод	Перехід до наступної сторінки згідно з хронологією
Navigate	метод	Перехід за вказаною URL-адресою
Navigating	подія	Настає перед виконанням переходу, що дозволяє скасувати дію
ObjectForScripting	властивість	Отримує або задає об'єкт, який може використовуватися кодом скрипту веб-сторінки для взаємодії з програмою
Print	метод	Друк поточної веб-сторінки
Refresh	метод	Перезавантаження поточної веб-сторінки
Stop	метод	Перериває поточну операцію переходу та зупиняє роботу динамічних елементів сторінки, таких як звук та анімація
Url	властивість	Отримує або вказує URL-адресу поточної веб-сторінки. При встановленні значення цієї властивості елемент керування переходить до нової URL-адреси

Приклад використання *WebBrowser*.

Приклад 57.

У лістингу 4.59 та рис. 4.91 наведено відповідно програму та результат її виконання, з використанням елемента керування *WebBrowser* для відображення *Flash-файлів*.

Лістинг 4.59. (examp57)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp57
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            this.webBrowser1.Dock = System.Windows.Forms.DockStyle.Fill;
            this.webBrowser1.Location = new System.Drawing.Point(0, 0);
            this.webBrowser1.MinimumSize = new System.Drawing.Size(20, 20);
            this.webBrowser1.Name = "webBrowser1";
            this.webBrowser1.Size = new System.Drawing.Size(647, 493);
            this.webBrowser1.TabIndex = 0;

            //   webBrowser1.Navigate("www.mail.ru");
            webBrowser1.Navigate(System.IO.Directory.GetCurrentDirectory() +
           "\\Shar.swf");
        }
    }
}
```

Рис. 4.91. Відображення Flash-файлів

Приклад 58.

У цьому прикладі наведено відображення в *WebBroser* таблиці, записаної на мові *HTML* за допомогою елементарних тегів *<tr>* (рядок таблиці) та *<td>* (комірка таблиці). Тут у форму встановлюється з *ToolBox* елемент керування *WebBroser*. Текст програми наведено у лістингу 4.60.

Лістинг 4.60. (examp58)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp58
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            // webBrowser1.Navigate("d:\\table.htm");
            string Str_html = "Какой-либо текст до таблицы" +
                                    "<table border> " +
                                    "<caption>Таблица телефонов</caption> " +
                                    "<tr><td>Андрей – раб<td>274-88-17 " +
                                    "<tr><td>Света-Х<td>+38(067)7030356 " +
                                    "<tr><td>ЖЭК<td>22-345-72 " +
                                    "<tr><td>Справка по тел<td>009 " +
                                    "</table> " +
                                    "Какой-либо текст после таблицы";
           //webBrowser1.Navigate("about:" + Str_html);
            webBrowser1.Navigate(Str_html);

        }
    }
 }
```

Як видно з тексту програми, у рядковій змінній *Str_html* задаємо *HTML-подання* найпростішої таблиці, що складається з двох стовпців та чотирьох рядків. Цей рядок подаємо на вхід методу *Navigate* об’єкта *webBrowser1*, прикріплюючи його до ключового слова "about:". У коментарі показано, як можна подати на вхід методу *Navigate* таблицю, якщо вона вже записана у файл *table.htm*. Результат роботи програми показано на рис. 4.92.

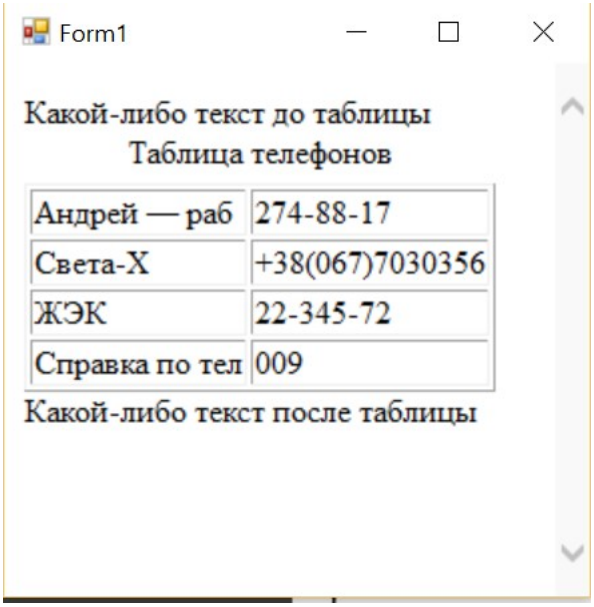


Рис. 4.92. Виведення таблиці у WebBroser

Приклад 59.

Завдання полягає в тому, щоб у *WebBroser* відобразити будь-яку *Web-сторінку*, що містить *Web-форму*. Потім програмно заповнити поля форми (наприклад, логін та пароль) на поштовому сервері або рядок пошуку у пошуковій системі. А при завершенні натиснути кнопку *Submit* (вона може називатися по-різному: *OK*, *Send* тощо), для надсилання заповненої форми для обробки на сервер.

Програма завантажує до елемента *WebBrowser* початкову сторінку пошукової системи *http://yahoo.com*. Далі використовуючи посилання на некерований інтерфейс *DomDocument* (властивість об'єкта класу *WebBrowser*), наводимо його до покажчика *IHTMLDocument2*. У цьому випадку ми отримуємо доступ до форм та полів *Web-сторінки* за їх іменами. Заповнюємо поле пошуку ключовими словами для знаходження відповідних *Web-сторінок*, а потім для відправлення заповненої форми на сервер програмно натискаємо кнопку *Submit*. У результаті отримаємо в елементі *WebBrowser* результат роботи пошукової системи, а саме безліч посилань на сторінки, що містять вказані ключові слова. У цю програму необхідно імпортувати некеровану бібліотеку *Microsoft.mshtml.dll*, для цього в пункті меню *Project* - вибираємо команду *Add Reference*, а на вкладці *.NET* двічі натисніть на посилання *Microsoft.mshtml*.

Програмний код наведено у лістингу 4.61.

Лістинг 4.61. (examp59)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace examp59
```

```
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            this.Text = "Программное заполнение формы";
            // Для сайта "http://google.com":
            //string АдресСайта = "http://google.com";
            //string ИмяФормы = "f";
            //string ИмяПоляФормы = "q";
            // Для сайта "http://meta.ua":
            //string АдресСайта = "http://meta.ua";
            //string ИмяФормы = "sForm";
            //string ИмяПоляФормы = "q";
            // Для сайта "http://yandex.ru":
            //string АдресСайта = "http://yandex.ru";
            //string ИмяФормы = "form";
            //string ИмяПоляФормы = "text";
            // Для сайта "http://rambler.ru":
            //string АдресСайта = "http://rambler.ru";
            //string ИмяФормы = "rSearch";
            //string ИмяПоляФормы = "query";
            // Для сайта "http://aport.ru":
            //string АдресСайта = "http://aport.ru";
            //string ИмяФормы = "aport_search";
            //string ИмяПоляФормы = "r";
            // Для сайта "http://bing.com":
            //string АдресСайта = "http://bing.com";
            //string ИмяФормы = "sb_form";
            // - в HTML-коде нет name формы, но есть id = "sb_form"
            //string ИмяПоляФормы = "q";
            // Для сайта "http://yahoo.com":
            string АдресСайта = "http://yahoo.com";
            string ИмяФормы = "sf1"; // или "p_13838465-searchform"
            string ИмяПоляФормы = "p"; // или "p_13838465-p"
            // Загружаем Web-документ в элемент WebBrowser:
            webBrowser1.Navigate(АдресСайта);
            while (webBrowser1.ReadyState != WebBrowserReadyState.Complete)
            {
                Application.DoEvents();
                System.Threading.Thread.Sleep(50);
            }
            if (webBrowser1.Document == null)
            {
              MessageBox.Show("ERROR: Возможно, вы не подключены к Интернет");
                return;
            }

            // Свойство DomDocument приводим к указателю IHTMLDocument2:
            mshtml.IHTMLDocument2 Док =
            (mshtml.IHTMLDocument2)webBrowser1.Document.DomDocument;
        // В этом случае мы получаем доступ к формам Web-страницы по их именам:
            mshtml.HTMLFormElement Форма =
            (mshtml.HTMLFormElement)Док.forms.item(ИмяФормы, null);
            if (Форма == null)
            {
                MessageBox.Show(String.Format("ERROR: Форма с " +
                "именем \"{0}\" не найдена", ИмяФормы));
                return;
            }
            // В форме находим нужное поле по его (полю) имени:
```

```
                mshtml.IHTMLInputElement ТекстовоеПоле =
                (mshtml.IHTMLInputElement) Форма.namedItem("p");
                if (ТекстовоеПоле == null)
                {
                    MessageBox.Show(String.Format("ERROR: Поле формы с " +
                    "именем \"{0}\" не найдено ", ИмяПоляФормы));
                    return;
                }
                // Заполняем текстовое поле:
                string query = "Савченко";
                ТекстовоеПоле.value = query;
                // "Программно" нажимаем кнопку "Search":
                Форма.submit();
            }
        }
    }
```

Як видно з програмного коду, відразу після завершення *InitializeComponent()* для пошукової системи *Yahoo* встановлюємо її адресу, ім'я форми та адресу форми. Для інших пошукових систем дані наводяться у коментарях.

Після первинного завантаження *Web-документа* елемент *WebBrowser*, використовуючи вказівник на некерований інтерфейс *DomDocument* (властивість об'єкта *WebBrowser*), наводимо його до покажчика *IHTMLDocument2*. У цьому випадку ми отримуємо доступ до форм та полів *Web-сторінки* за їх іменами. Далі заповнюємо поле пошуку ключовими словами для знаходження посилань на відповідні *Web-сторінки*, а потім для надсилання заповненої форми на сервер натискаємо кнопку *Submit*. В результаті отримаємо в елементі *WebBrowser* результат роботи пошукової системи, як показано на рис. 4.93.

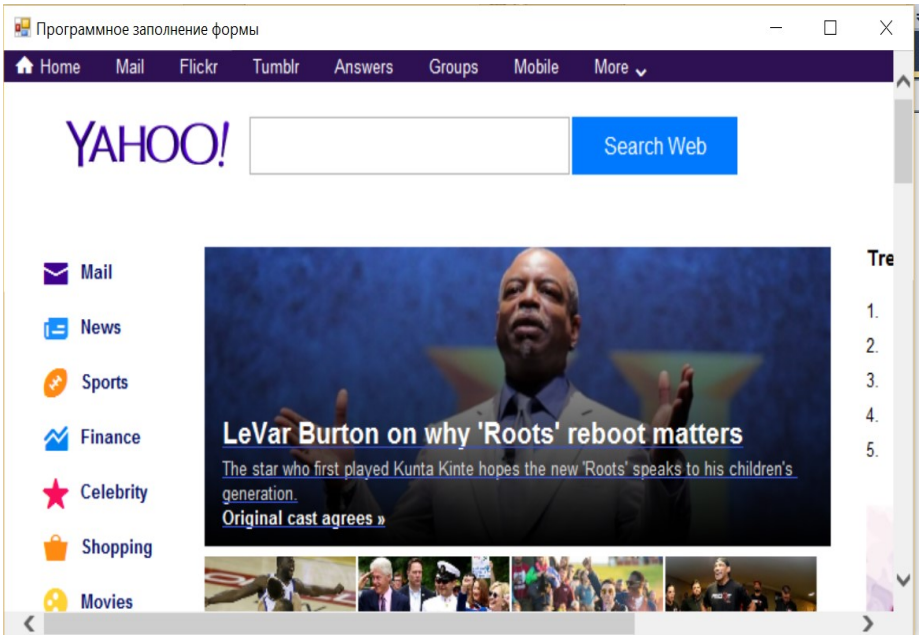


Рис. 4.93. Результат пошукової системи

4.27. Елемент керування LinkLabel

Елемент керування *LinkLabel* може використовуватися для тих самих цілей, що й елемент керування *Label*. Можна встановити частину тексту як посилання на об'єкт або веб-сторінку. *LinkLabel* дозволяє створити посилання у стилі веб. Крім того, можна задати частину тексту як посилання на файл, папку або *веб-сторінку*. При виконанні натискання на посилання є можливість зміни кольору, щоб вказати, що до цього посилання вже зверталися.

Додатково до всіх властивостей, методів і подій елемента керування *Label*, елемент керування *LinkLabel* має ще властивості для гіперпосилань і кольору посилань. Властивість *LinkArea* визначає область тексту, яка активує посилання.

Також, як і зі звичайними посиланнями на веб-сторінках, ми можемо по відношенню до цього елемента визначити три кольори:

- Властивість *ActiveLinkColor* задає колір посилання при натисканні
- Властивість *LinkColor* задає колір посилання до натискання, за яким не було переходів
- Властивість *VisitedLinkColor* задає колір посилання, за яким вже були переходи

Крім кольору посилання для даного елемента ми можемо задати властивість *LinkBehavior*, яка керує поведінкою посилання. Ця властивість набуває чотирьох можливих значень:

- *SystemDefault*: для посилання встановлюються системні налаштування
- *AlwaysUnderline*: посилання завжди підкреслюється
- *HoverUnderline*: посилання підкреслюється лише при наведенні на нього курсору миші
- *NeverUnderline*: посилання ніколи не підкреслюється

Щоб виконати перехід за посиланням при натисканні на нього, потрібно додатково написати код. Цей код повинен обробляти подію *LinkClicked*, яка має елемент *LinkLabel*.

Найпростіше застосування елемента керування *LinkLabel* - відображення одиночного посилання з використанням властивості *LinkArea*, проте також можливе відображення кількох гіперпосилань за допомогою властивості *Links*. Властивість *Links* дозволяє отримати доступ до колекції посилань. Крім того, як *LinkLabel.Link* можна визначити дані для кожного окремого об’єкта *LinkData*. Значення властивості *LinkData* служить для зберігання розміщення файлу, що відображається, або адреси *веб-вузла*.

Розглянемо два приклади використання *LinkLabel*.

Приклад 60.

Цей приклад досить простий. Тут виконується посилання на веб-сторінку *www.microsoft.com* та її виведення. Текс програми наведено у лістингу 4.62. Відображення посилання та виведення *Web-сторінки* наведено на рис. 4.94. та рис. 4.95 відповідно.

Лістинг 4.62. (examp60)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp60
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            this.linkLabel1.AutoSize = true;
            this.linkLabel1.Text = "Visit Microsoft";
            this.Text = "Simple Link Label Example";
        }
        private void linkLabel1_LinkClicked
        (object sender, System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
        {
          this.linkLabel1.LinkVisited = true;
          System.Diagnostics.Process.Start("http://www.microsoft.com");
        }
    }
}
```

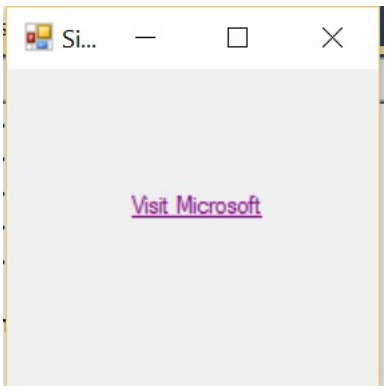


Рис. 4.94. До використання *LinkLabel*

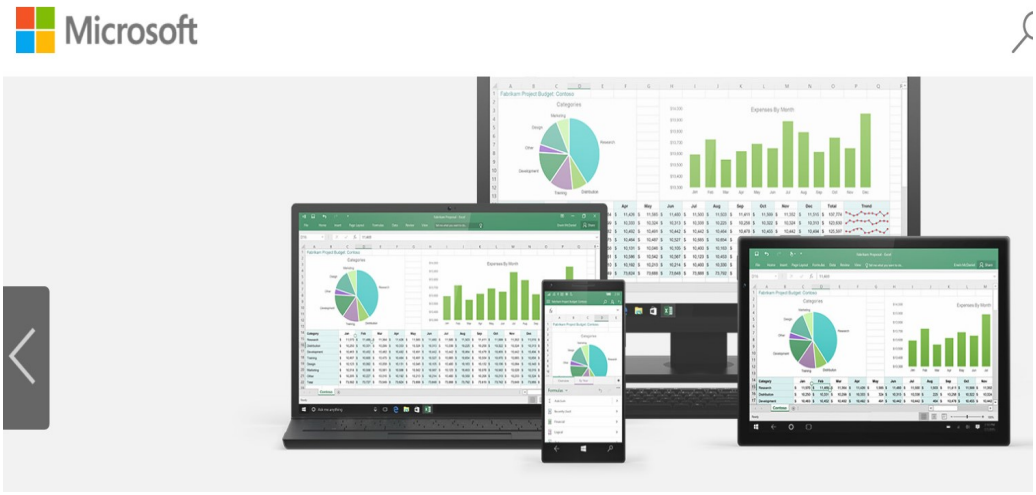


Рис. 4.95. Web-сторінка `www.microsoft.com`.

Приклад 61.

У цьому прикладі, програму якого наведено на лістингу 4.63, демонструється використання *LinkLabel* з декількома *LinkArea* розділами, визначені для відображення мітки у формі. Властивості для налаштування зовнішнього вигляду *LinkLabel*. Перше *LinkArea* задається як об'єкт класу *LinkLabel. LinkArea*. Додаткові посилання додаються до *LinkLabel* за допомогою методу *LinkLabel1.Links.Add*. У прикладі показано обробку події *LinkClicked*. Тут виконується запуск *веб-браузера* для гіперпосилань та підключення *MessageBox* для інших посилань.

Лістинг 4.63. (examp61)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp61
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            this.linkLabel1.AutoSize = true;
            this.linkLabel1.Location = new System.Drawing.Point(28, 55);
            this.linkLabel1.Name = "linkLabel1";
            this.linkLabel1.Size = new System.Drawing.Size(80, 20);
            this.linkLabel1.TabIndex = 0;
            this.linkLabel1.TabStop = true;
            this.linkLabel1.Text = "linkLabel1";
            this.linkLabel1.LinkClicked +=
            new System.Windows.Forms.LinkLabelLinkClickedEventHandler
            (this.linkLabel1_LinkClicked);
```

```
            this.linkLabel1.DisabledLinkColor = System.Drawing.Color.Red;
            this.linkLabel1.VisitedLinkColor = System.Drawing.Color.Blue;
            this.linkLabel1.LinkBehavior =
            System.Windows.Forms.LinkBehavior.HoverUnderline;
            this.linkLabel1.LinkColor = System.Drawing.Color.Navy;
            this.linkLabel1.TabIndex = 0;
            this.linkLabel1.TabStop = true;

            // Идентификация первой строки
            this.linkLabel1.LinkArea = new LinkArea(0, 8);

            // Identify that the first link is visited already.
            this.linkLabel1.Links[0].Visited = true;
            this.linkLabel1.Text = "Register Online.  Visit Microsoft.  Visit
MSN.";
            if (this.linkLabel1.Text.Length >= 45)
            {
                //this.linkLabel1.Links[0].LinkData = "Register";
                this.linkLabel1.Links.Add(24, 9, "www. google.ru");
                this.linkLabel1.Links.Add(42, 3, "www. msn.com");
                this.linkLabel1.Links[1].Enabled = false;
            }
        }

        private void linkLabel1_LinkClicked(object sender,
        System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
        {
            string target = e.Link.LinkData as string;
            //Если значение target выглядит как URL , активизируем страницу.
            // В противном случае , target отобразится в окне сообщения .
            if (null != target && target.StartsWith("www"))
            {
                System.Diagnostics.Process.Start(target);
            }
            else
            {
                MessageBox.Show("Item clicked: " + target);
            }
        }
    }
}
```

Як видно з рис. 4.96. при виконанні програми рядок гіперпосилань виводиться у формі. Тут синім кольором виводяться два гіперпосилання: "Register" та "MSN". При натисканні "миші" на цих гіперпосиланнях у першому випадку результат виводиться в *MessageBox*, у другому випадку відкривається *Web-сторінка* (див. рис. 4.97.) за встановленою *URL-адресою* "www.msn.com". Червоним кольором виділено гіперпосилання "www.google.ru", яке не є активним і ніяких дій при його натисканні не відбувається. Колір цього гіперпосилання встановлюється властивістю

```
linkLabel1.DisabledLinkColor = System.Drawing.Color.Red
```

Гіперпосилання стає не активним після встановлення властивості

```
this.linkLabel1.Links[1].Enabled = false;
```

Перші два числові параметри методів *LinkArea* і *Links.Add* визначають межі виділення гіперпосилання із загального рядка, що визначається властивістю *linkLabel1.Text*. Тут перший параметр визначає номер символу в загальному рядку, другий довжину гіперпосилання

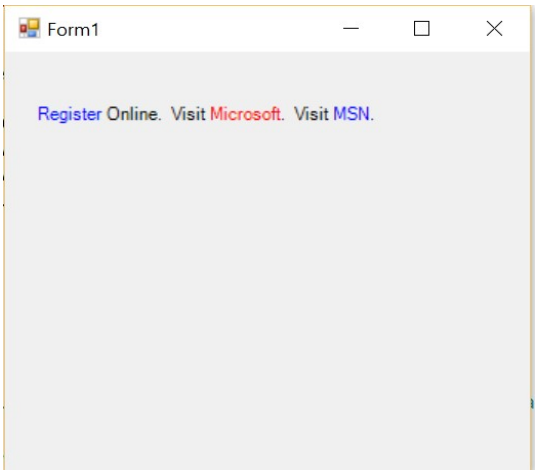


Рис. 4.96. Відображення гіперпосилання у *LinkLabel*

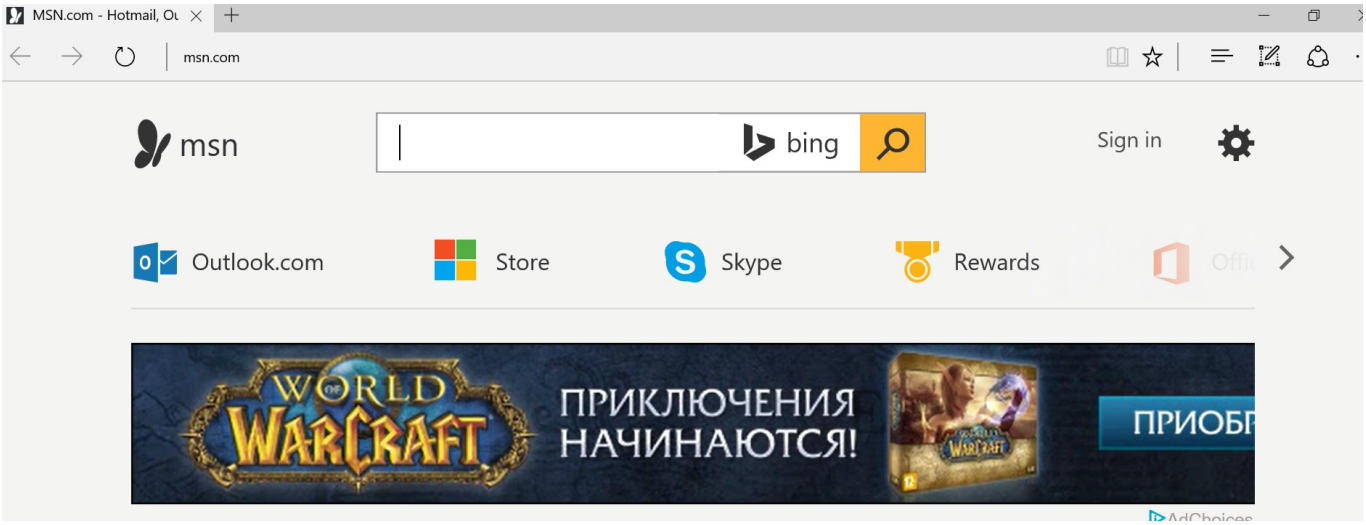


Рис. 4.97. Web-сторінка `www.msn.com`

4.28. Робота з реєстром

При використанні стандартних діалогів недоцільно під час кожного виконання програми вибирати параметри цих діалогів. Очевидно, що вибрані кольори, шрифти, розміри вікон і т.д. доцільно зберігати у файлах (*ini-файлах*) чи реєстрах. Збережені дані завантажуються спочатку програми та зберігаються після її закриття. При цьому дані слід зберігати в тому випадку, якщо вони були змінені під час виконання програми. Розглянемо роботу з реєстром.

Для роботи з реєстром використовується клас *RegistryKey*. Цей клас надає набір стандартних кореневих розділів, що знаходяться в реєстрі комп'ютерів, які працюють під керуванням *Windows*.

Базові (кореневі) екземпляри *RegistryKey*, які показуються класом *Registry*, створюють схему базового механізму зберігання у реєстрі вкладених розділів та значень. Усі розділи доступні лише для читання, оскільки реєстр залежить від їх існування. Клас *Registry* надає доступ до наступних розділів:

CurrentUser - зберігає відомості про параметри користувача.

LocalMachine - зберігає інформацію про конфігурації для локального комп'ютера.

ClassesRoot - зберігає відомості про типи (і класи) та їх властивості.

Users - зберігає відомості про стандартну конфігурацію користувача.

PerformanceData - зберігає інформацію про продуктивність програмних компонентів.

CurrentConfig - зберігає відомості про обладнання, яке не є специфічним для користувача.

DynData – зберігає динамічні дані.

Визначивши кореневий розділ реєстру, в якому необхідно зберігати та з якого необхідно отримувати відомості, можна використовувати клас *RegistryKey*, щоб додавати або видаляти вкладені розділи та виконувати дії зі значеннями цього розділу.

Апаратні пристрої можуть розміщувати відомості в реєстрі автоматично, використовуючи інтерфейс *Plug and Play*. Програмні засоби встановлення драйверів пристрою можуть розміщувати відомості в реєстрі шляхом запису в стандартні інтерфейси *API*.

У платформі *.NET Framework* версії 2.0 клас *Registry* також містить методи *GetValue* і *SetValue* для визначення та отримання значень розділів реєстру. класу *RegistryKey*.

Клас *RegistryKey* також надає методи, що дозволяють задати безпеку елементів керування доступом для розділів реєстру, щоб перевірити тип даних значення перед його поверненням та видалення ключів.

У таблиці 4.8. наведено основні методи та поля класу *RegistryKey*

Таблиця 4.8

Методи та поля класу *RegistryKey*

Им'я	**Опис**
Методи	
GetValue(String, String, Object)	Повертає значення, пов'язане зі зазначеним ім'ям, у вказаному розділі реєстру. Якщо ім'я не знайдено у вказаному розділі, повертає значення за замовчуванням або значення null, якщо вказаний розділ не існує.
SetValue(String, String, Object)	Вказує значення пари "ім'я-значення" для зазначеного розділу реєстру. Якщо цей розділ не існує, він буде створений.
SetValue(String, String, Object, RegistryValueKind)	Задає пару "ім'я-значення" для вказаного розділу реєстру, використовуючи вказаний тип даних реєстру. Якщо цей розділ не існує, він буде створений.
ClassesRoot	Визначає типи (або класи) документів та властивості, пов'язані з цими типами. Це поле зчитує базовий розділ реєстру Windows HKEY_CLASSES_ROOT.

Продовження таблиці 4.8

Им’я	Опис
Поля	
CurrentConfig	Містить відомості про конфігурацію, що стосуються обладнання, не пов’язаного з конкретним користувачем. Це поле зчитує базовий розділ реєстру Windows HKEY_CURRENT_CONFIG.
CurrentUser	Містить відомості про поточні параметри користувача. Це поле зчитує базовий розділ реєстру Windows HKEY_CURRENT_USER.
DynData	**Застаріло**. Містить динамічні дані реєстру. Це поле зчитує базовий розділ реєстру Windows **HKEY_DYN_DATA**.
LocalMachine	Містить дані про конфігурацію локального комп’ютера. Це поле зчитує базовий розділ реєстру Windows HKEY_LOCAL_MACHINE.
PerformanceData	Містить дані про продуктивність компонентів програмного забезпечення. Це поле зчитує базовий розділ реєстру Windows HKEY_PERFORMANCE_DATA.
Users	Містить відомості про стандартну конфігурацію користувача. Це поле зчитує базовий розділ реєстру Windows HKEY_USERS.

Розглянемо чотири приклади використання реєстру. У першому прикладі демонструються кореневі розділи, на другому - методи *static GetValue* і *SetValue*. У третьому та четвертому прикладах показано збереження кольору, шрифтів, параметрів форми та її розташування.

Приклад 62.

Тут показано спосіб отримання вкладених розділів (підрозділів) розділу *HKEY_USERS* та виведення їх на екран. Текст програми та виведення найменувань підрозділів наведено відповідно на лістингу 4.64. та рис. 4.98.

Лістинг 4.64. (examp62)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Microsoft.Win32;

namespace examp62
{
    public partial class Form1 : Form
    {
        public string str;
        public Form1()
        {
            InitializeComponent();
            RegistryKey rk = Registry.Users;
            PrintKeys(rk);
```

```
        }

        void PrintKeys(RegistryKey rkey)
        {
            str = "";
            string[] names = rkey.GetSubKeyNames();
            int icount = 0;
            str += "Subkeys of  " + rkey.Name;
            str+= "\n---------------------\n";
            foreach (string s in names)
            {
                str += s + "\n";
                icount++;
                if (icount >= 10) break;
            }
            MessageBox.Show(str);
        }
    }
}
```

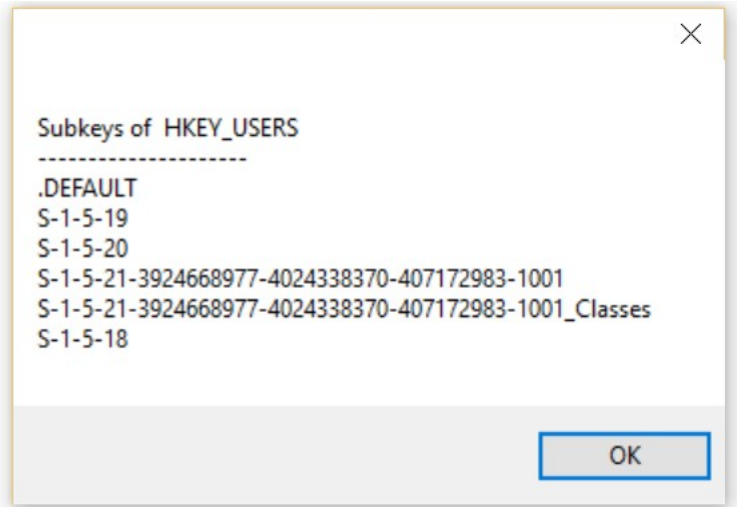


Рис. 4.98. Виведення підрозділів розділу *HKEY_USERS*

Приклад 63.

Тут створюється розділ реєстру, в якому зберігаються значення кількох типів даних, а потім виконується повернення та відображення значень. У цьому прикладі показано збереження та повернення стандартних (безіменних) пар "ім'я-значення", а також використання *defaultValue*, якщо пара "ім'я-значення" не існує. Текст програми наведено у лістингу 4.65. Виведення проміжних даних наведено на рис. 4.99.

Лістинг 4.65. (examp63)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Microsoft.Win32; // Registry

namespace examp63
```

```
{
    public partial class Form1 : Form
    {
        const string userRoot = "HKEY_CURRENT_USER";
        const string subkey = "RegistrySetValueExample";
        const string keyName = userRoot + "\\" + subkey;
        string str;
        public Form1()
        {
            InitializeComponent();
            str = "";
            //Значение INT может храниться без указания типа данных реестра.
            //Тип Long сохраняется в виде строки, если этот тип не указан.
            // INT хранится в паре имя / значение по умолчанию.
            Registry.SetValue(keyName, "", 5280);
            Registry.SetValue(keyName, "TestLong", 12345678901234,
.....RegistryValueKind.QWord);
           // Строки с возможностью расширения переменных среды хранятся в виде
           //  обычных строк, если не указан тип данных.
            Registry.SetValue(keyName, "TestExpand", "My path: %path%");
            Registry.SetValue(keyName, "TestExpand2", "My path: %path%",
            RegistryValueKind.ExpandString);
      //Массивы строк автоматически сохраняются в виде многостраничного String.
            string[] strings = { "One", "Two", "Three" };
            Registry.SetValue(keyName, "TestArray", strings);
            string noSuch = (string)Registry.GetValue(keyName,
           "NoSuchName", "Return this default if NoSuchName does not exist.");
            str += "SuchName: " + noSuch + "\n";
            int tInteger = (int)Registry.GetValue(keyName, "", -1);
            str += "Default: " + tInteger.ToString() + "\n";
            long tLong = (long)Registry.GetValue(keyName, "TestLong",
            long.MinValue);
            str += "TestLong: " + tLong + "\n";
            string[] tArray = (string[])Registry.GetValue(keyName,
            "TestArray",
            new string[] { "Default if TestArray does not exist." });
            for (int i = 0; i < tArray.Length; i++)
            {
                str += i.ToString() + "  " + tArray[i] + "\n";
            }

            string tExpand = (string)Registry.GetValue(keyName,
            "TestExpand",
            "Default if TestExpand does not exist.");
            str += tExpand + "\n";
            string tExpand2 = (string)Registry.GetValue(keyName,
            "TestExpand2",
            "Default if TestExpand2 does not exist.");
            Console.WriteLine("TestExpand2: {0}...",
            tExpand2.Substring(0, 40));
            str += "TestExpand2: " + tExpand2.Substring(0, 40) + "\n";
            str += " nUse the registry editor to examine the key. \n";
            Console.WriteLine("Press the Enter key to delete the key. \n");
            str += " Press the Enter key to delete the key.";
            Console.ReadLine();
            Registry.CurrentUser.DeleteSubKey(subkey);
            MessageBox.Show(str);
        }
    }
}
```

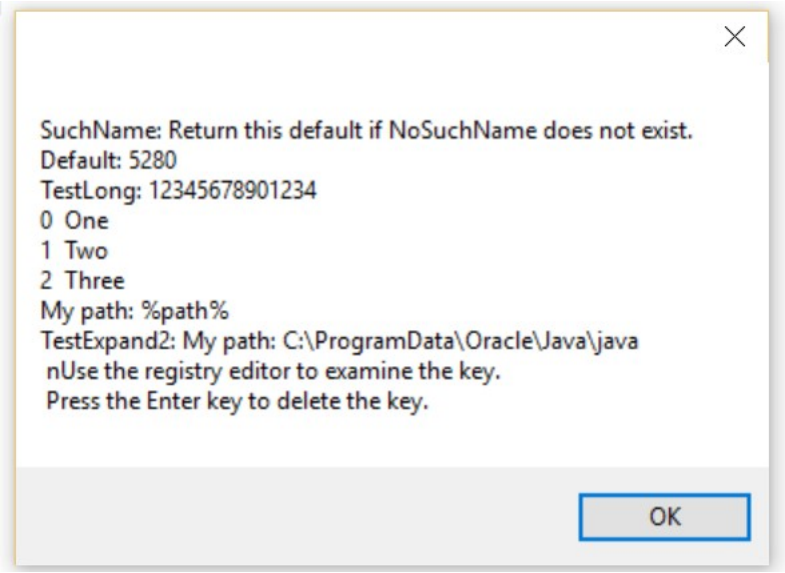


Рис. 4.99. Виведення проміжних результатів у програмі

Приклад 64.

Дані, отримані стандартними діалогами вибору кольорів і шрифтів записуються в реєстр. При кожному завантаженні програма ці дані зчитуються з реєстру і в *Label* виводиться слово "Привіт" з раніше збереженими властивостями шрифту (ім'я шрифту, розмір та стиль) та кольори переднього плану та фону. Текст програми наведено у лістингу 4.66. Фрагмент виконання програми наведено на рис. 4.100.

Лістинг 4.66. (examp64)

```
using System.Text;
using System.Windows.Forms;
using Microsoft.Win32;

namespace examp64
{
    //Работа со стандартными диалогами с сохранением данных в реестр
    public partial class Form1 : Form
    {
        const string strRegKey = "Software\\ProgrammingCSharp\\Reg";
        const string strFontFace = "FontFace";
        const string strFontSize = "FontSize";
        const string strFontStyle = "FontStyle";
        const string strForeColor = "ForeColor";
        const string strBackColor = "BackColor";

        public Form1()
        {
            InitializeComponent();
            // Считывание данных с реестра
            RegistryKey regkey = Registry.CurrentUser.OpenSubKey(strRegKey);
            if (regkey != null)
            {
                Font ff = new Font((string)regkey.GetValue(strFontFace),
                float.Parse((string)regkey.GetValue(strFontSize)),
                (FontStyle)regkey.GetValue(strFontStyle));
                label1.ForeColor =
Color.FromArgb((int)regkey.GetValue(strForeColor));
```

```
                    label1.BackColor =
Color.FromArgb((int)regkey.GetValue(strBackColor));
                    label1.Font = ff;
                    label1.Text = "Привет";
                    regkey.Close();
                }
            }

            protected override void OnClosed(EventArgs e)
            {
                // Запись данных в реестр
                RegistryKey regkey = Registry.CurrentUser.OpenSubKey(strRegKey,
true);
                if (regkey == null)
                regkey = Registry.CurrentUser.CreateSubKey(strRegKey);

                regkey.SetValue(strFontFace, label1.Font.Name);
                regkey.SetValue(strFontSize, label1.Font.SizeInPoints.ToString());
                regkey.SetValue(strFontStyle, (int)label1.Font.Style);
                regkey.SetValue(strForeColor, label1.ForeColor.ToArgb());
                regkey.SetValue(strBackColor, label1.BackColor.ToArgb());
                regkey.Close();
                base.OnClosed(e);
            }

            private void button1_Click(object sender, EventArgs e)
            {
                fontDialog1.Font = label1.Font;
                if (fontDialog1.ShowDialog() == DialogResult.OK)
                label1.Font = fontDialog1.Font;
            }

            private void button2_Click(object sender, EventArgs e)
            {
                colorDialog1.Color = label1.ForeColor;
                if (colorDialog1.ShowDialog() == DialogResult.OK)
                    label1.ForeColor = colorDialog1.Color;
            }
        }
    }
```

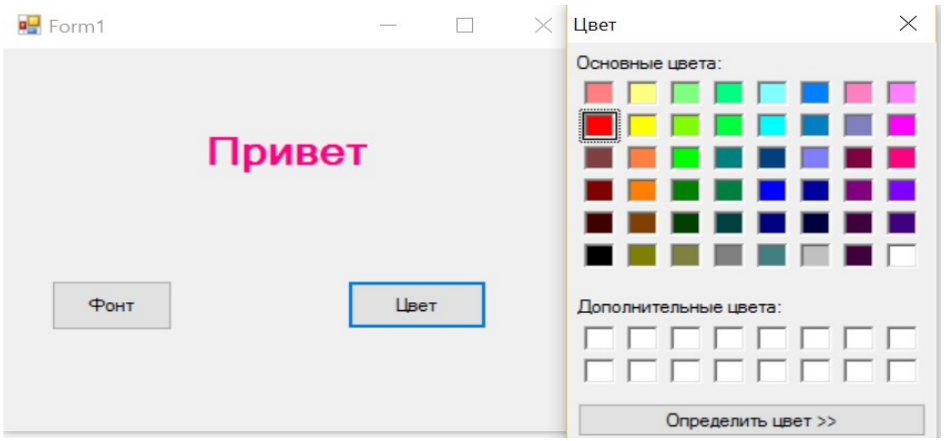


Рис. 4.100. Обрання кольору

Приклад 65.

Тут у реєстрі зберігається розташування вікна (координати лівого верхнього кута), його ширина та висота. Ці параметри зберігаються під час закриття форми та використовуються під час її завантаження. Текст програми наводиться у лістингу 4.67. Результат виконання програми представлено на рис. 4.101. Тут показано форму з параметрами, збереженими у реєстрі.

Лістинг 4.67. (examp65)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Microsoft.Win32;

namespace examp65

    //Работа с реестром
    public partial class Form1 : Form
    {
        Rectangle rectNormal;
        protected string strProgName;
        string strRegKey = "Softwar\\ProgrammingWindowsWithCSharp\\";
        const string strWinState = "WindowState";
        const string strLocationX = "LocationX";
        const string strLocationY = "LocationY";
        const string strWidth = "Width";
        string strHeight = "Height";

        public Form1()
        {
            InitializeComponent();
            Text = "Notepad Clone No";

            textBox1.Dock = DockStyle.Fill;
            textBox1.BorderStyle = BorderStyle.None;
            textBox1.Multiline = true;
            textBox1.ScrollBars = ScrollBars.Both;
            textBox1.AcceptsTab = true;
            Text = strProgName = "Notepad Clone with Registry";
            rectNormal = this.DesktopBounds;
        }

        private void Form1_Move(object sender, EventArgs e)
        {
            if (WindowState == FormWindowState.Normal)
                rectNormal = DesktopBounds;
        }

        private void Form1_Resize(object sender, EventArgs e)
        {
            if (WindowState == FormWindowState.Normal)
                rectNormal = DesktopBounds;
        }

        //Загрузка формы
```

```
        private void Form1_Load(object sender, EventArgs e)
        {

            // Формируем полный раздел реестра.
            strRegKey = strRegKey + strProgName;
            // Загружаем информацию из реестра.
            try
            {
                RegistryKey regkey = null;
                regkey = Registry.CurrentUser.OpenSubKey(strRegKey, true);
                if (regkey != null)
                    LoadRegistryInfo(regkey);
                regkey.Close();
            }
            catch (Exception)
            {
                return;
            }
        }

        //Закрытие формы
        private void Form1_FormClosed(object sender, FormClosedEventArgs e)
        {
            RegistryKey regkey =
            Registry.CurrentUser.OpenSubKey(strRegKey, true);
            if (regkey == null)
                regkey = Registry.CurrentUser.CreateSubKey(strRegKey);
            SaveRegistryInfo(regkey);
            regkey.Close();
        }

        //Чтение данных из реестра
        protected virtual void LoadRegistryInfo(RegistryKey regkey)
        {
            int x =  (int) regkey.GetValue(strLocationX, 100);
            int y =  (int) regkey.GetValue(strLocationY, 100);
            int cx = (int) regkey.GetValue(strWidth, 300);
            int cy = (int) regkey.GetValue(strHeight, 300);

            Location = new Point(x, y);
            Size = new Size(cx, cy);
            WindowState = (FormWindowState)regkey.GetValue
            (strWinState, FormWindowState.Normal);
            //Text = cx.ToString();
        }

        //Запись данных в реестр
        protected virtual void SaveRegistryInfo(RegistryKey regkey)
        {
            regkey.SetValue(strWinState, (int) WindowState);
            regkey.SetValue(strLocationX, rectNormal.X);
            regkey.SetValue(strLocationY, rectNormal.Y);
            regkey.SetValue(strWidth, rectNormal.Width);
            regkey.SetValue(strHeight, rectNormal.Height);
        }
    }
}
```

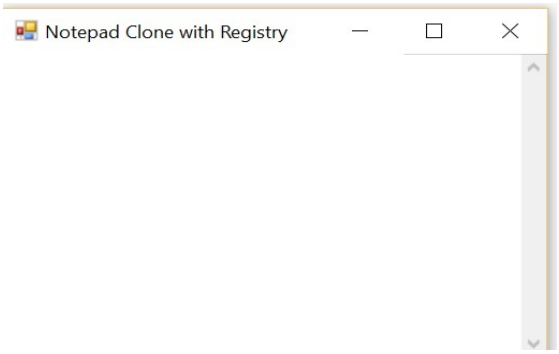


Рис. 4.101. Параметри форми за даними реєстру

5. Завантаження та виведення зображення

5.1. Виведення зображення з файлу та Інтернету

Якщо потрібно лише завантажувати та виводити растрові зображення, клас *Image* містить все необхідне для цього. Клас *Image* розташований у просторі імен *System.Drawing*. Цей клас є абстрактним класом, який надає функціональні можливості для похідних класів Bitmap та Metafile.

Формати файлів, що підтримуються класом *Image*, вказуються у статичних властивостях класу *ImageFormat*, визначеного у просторі імен *System.Drawing.Imaging*: *bmp*, *MemoryBmp*, *Icon*, *Gif*, *Jpeg*, *Png*, *Tiff*, *Exif*, *Wmf* та *Emf*.

Клас *Image* має 4 статичні методи, які повертають об’єкти типу *Image* та необхідні для завантаження бітової карти або метафайлу з файлу або потоку:

```
1. public static Image FromFile(  string filename );
//string filename - //шлях до файлу, що завантажується

2. public static Image FromFile(string filename, bool useEmbeddedColorManagement);
// useEmbeddedColorManagement  - параметр, який вказує чи потрібно
// використовувати інформацію про налаштування кольору

3. public static Image FromStream( Stream stream);
// Stream stream – поток, що завантажується

4. public static Image FromStream( Stream stream, bool
useEmbeddedColorManagement);
```

Статичний метод *Image FromStream* корисний при отриманні потоку джерела, відмінного від файлової системи. Наприклад, метод *FromStream* дозволяє завантажити зображення з Інтернету.

Клас *Image* включає властивості, серед яких є властивості, що вказують розмір зображення пікселів.

```
public Size Size {get;}
public int Height {get;}
public int Width {get;}
```

Об’єкт Image можна вивести на екран або принтер за допомогою метода *DrawImage* класса *Graphics*.

```
  public void DrawImage(
  Image image,
  Point point
);

public void DrawImage(
  Image image,
  Point[] destPoints  //массив из 3 структур Point, определяющих параллелограмм.
);

public void DrawImage(
Image image,
PointF point );
public void DrawImage(
Image image,
Rectangle rect) ;  // Rectangle - определение расположения и размера изображения
```

У наступному прикладі код призначений для роботи з *Windows Forms*. Обробник для події Paint. Об’єкт Graphics передається в подію та використовується для малювання зображення у формі.

Код виконує такі дії:

- Створює малюнок із файлу з ім’ям *SampImag.jpg*. Цей файл повинен знаходитися в тій же папці, що і файл програми, що виконується.
- Створює точку, в якій розміщуватиметься верхній лівий кут зображення.

```
protected override void OnPaint(PaintEventArgs e)
{
      Image newImage = Image.FromFile( "SampImag.jpg" );
      Point ulCorner = Point(100,100);
      e->Graphics->DrawImage( newImage, ulCorner );
}
```

Приклад 66.

У лістингу 5.1 наводиться приклад завантаження зображення з файлу та Інтернету. Зокрема, наводиться завантаження зображення з Інтернету під час використання методу *FromStream*. Завантаження з файлу під час використання методу *FromFile* закоментовано.

Лістинг 5.1. (examp66)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.IO;

namespace exam66
{
   public partial class Form1 : Form
   {
      Image image;
      public Form1()
      {
         InitializeComponent();
         string strURL =
         "http://www.kryvyirih.dp.ua/ua/osximage/name/251109545166015_n_8o";
          string strFile = "1.jpg";
         WebRequest webreq = WebRequest.Create(strURL);
         WebResponse webres = webreq.GetResponse();
         Stream stream = webres.GetResponseStream();
         image = Image.FromStream(stream);
         //image = Image.FromFile(strFile);
         ResizeRedraw = true;
         stream.Close();
       }
      protected override void OnPaint(PaintEventArgs e)
      {
         base.OnPaint(e);
         Graphics gr = e.Graphics;
         float cx = image.Width;
         float cy = image.Height;
         Rectangle rect = ClientRectangle;
         gr.DrawImage(image, (rect.Width - cx) / 2, (rect.Height - cy) / 2,
         image.Width, image.Height);
         gr.DrawLine(new Pen(Color.FromArgb(255, 0, 0)), 0, rect.Height / 2, rect.Width,
         rect.Height / 2);
      }
   }
}
```

Виведення зображення наведено на рис. 5.1.

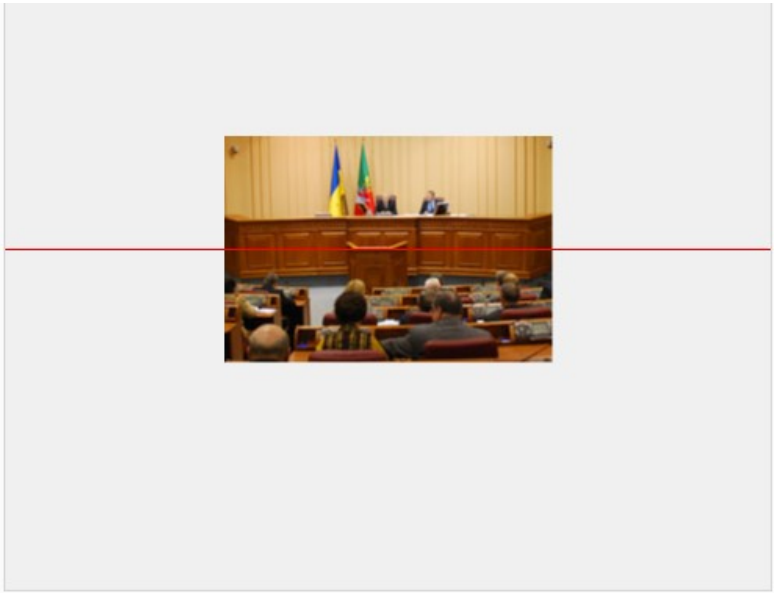


Рис. 5.1. Виведення зображення

Оператори, що використовують класи *WebRequest* та *WebResponse*, демонструють стандартний підхід до завантаження файлів з *Web*. У цій програмі метод *GetResponseStream* класу *WebResponse* отримує доступний для читання потік *JPEG-файлу*. На цьому етапі потік можна легко передати методу *FromStream* (*stream*). Для оцінки центрування зображення було проведено горизонтальну лінію.

5.2. Керування розміщенням зображення відносно клієнтського вікна

Версії *DrawImage*, орієнтовані на прямокутник, можуть масштабувати зображення, а й дещо інше. Якщо вказати негативну ширину, зображення буде повернено по вертикальній осі, і ви отримаєте її дзеркальну копію. При негативній висоті метод повертає її по горизонтальній осі і відображає догори ногами. У будь-якому випадку верхній лівий кут вихідного, неперевернутого зображення позиціонується згідно з координатами *Point* і *PointF* прямокутника, вказаними в *DrawImage*.

Приклад 67.

Програма на лістингу 5.2 виводить чотири зображення (див. рис. 5.2). Крім першого зображення, решта має негативну ширину або висоту. У всіх чотирьох випадках другий та третій аргументи методу *Draw Image* вказують центр клієнтської області.

Лістинг 5.2.(examp67)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.Linq;
using System.Text;
```

```
using System.Windows.Forms;
using System.Net;
using System.IO;

namespace examp67
{

   public partial class Form1 : Form
   {
      Image image;
      public Form1()
      {
         InitializeComponent();
         this.Size = new Size(1000, 300);
         Text = this.Size.ToString();
         string strURL =
         "http://www.kryvyirih.dp.ua/ua/osximage/name/251109545166015_n_8o";
         string strFile = "1.jpg";
         //WebRequest webreq = WebRequest.Create(strURL);
         //WebResponse webres = webreq.GetResponse();
         //Stream stream = webres.GetResponseStream();
         //image = Image.FromStream(stream);

         image = Image.FromFile(strFile);
         ResizeRedraw = true;
         //stream.Close();
      }

      protected override void OnPaint(PaintEventArgs e)
      {
         base.OnPaint(e);
         Graphics gr = e.Graphics;
         int cx_im = image. Width;
         int cy_im = image.Height;
         int wth = 10;
         gr.DrawImage(image,10,10,cx_im, cy_im);
         wth+=2*cx_im + 10;
         gr.DrawImage(image, wth, 10, -cx_im, cy_im);
         wth += 10;

         gr.DrawImage(image, wth, cy_im+10, cx_im, -cy_im);
         wth += 2*cx_im + 10;

         gr.DrawImage(image, wth, cy_im + 10, -cx_im, -cy_im);

      }
   }
}
```

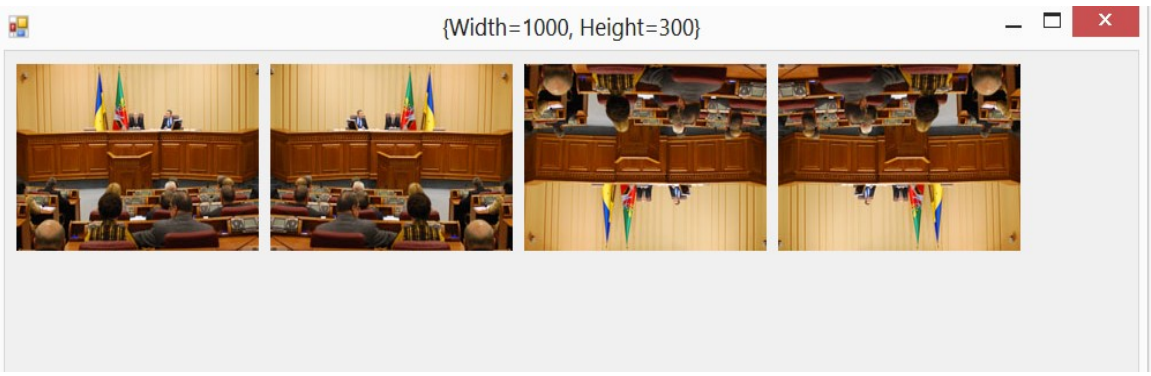


Рис. 5.2. До керування розміщенням зображення

5.3. Зміна розміру зображення

Клас *Image* включає кілька додаткових методів, що надають обмежені можливості збереження та маніпуляції з зображеннями.

Методи *Save* класу *Image* (вибірково):

void Save(strin g strFilename, ImageFormat if)
void Save(Stream stream , imageFormat if)

Метод *Save* не працює з об'єктами *Image*, завантаженими з метафайлу або що зберігаються у бітовій карті в пам'яті. Крім того, не можна зберегти зображення у форматі метафайлу або бітової картки пам'яті. Наступні два методи надають можливості зміни розміру, а також обертання та перевороту зображення відповідно

Методи класу *Image* (вибірково):

Image GetThumbnailImage(int ex, int cy, Image.GetThumbnaillmageAbort gtia, IntPtr pData);
void RotateFlip(RotateFliptype rft).

Метод *GetTbumbnaillmage* створює піктограми - зменшені зображення, що дозволяють програмам представляти їх вміст користувачу, заощаджуючи час і місце. Проте метод *GetTbumbnaillmage* фактично є універсальною функцією зміни розміру зображення. Зображення можна як збільшити, так і зменшити. Останні два аргументи вказують на функцію зворотного виклику, але їм можна задати значення *null* і *0*, що не позначиться на роботі методу.

Приклад 68.

Програму прикладу наведено у лістингу 5.3. Вона створює піктограму, що розміщується у квадраті зі стороною 64 пікселі. Методом *OnPaint()* вся клієнтська область заповнюється піктограмами (див. рис. 5.3).

Лістинг 5.3.(examp68)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp68
{
public partial class Form1 : Form
{
   const int isq = 64;
```

```
   Image image1;
  public Form1()
 {
InitializeComponent();
 ResizeRedraw = true;
Image image2 = Image.FromFile("1.jpg");
int cx, cy;
if (image2.Width > image2.Height)
{
   cx = isq;
   cy = cx* image2.Height / image2. Width;

}
else
{
   cy = isq;
   cx = cy *image2. Width/image2.Height;
}
   image1 = image2.G
}

protected override void OnPaint(PaintEventArgs e)
{
   base.OnPaint(e);
   int cxx = ClientSize.Width;
   int cyy = ClientSize.Height;
   for (int y = 0; y < cyy; y += isq)
   for (int x = 0; x < cxx; x += isq)
   e.Graphics.DrawImage(image1,
   x + (isq - image1.Width) / 2,
   y + (isq - image1.Height) / 2,
   image1.Width, image1.Height);
  }
 }}
```

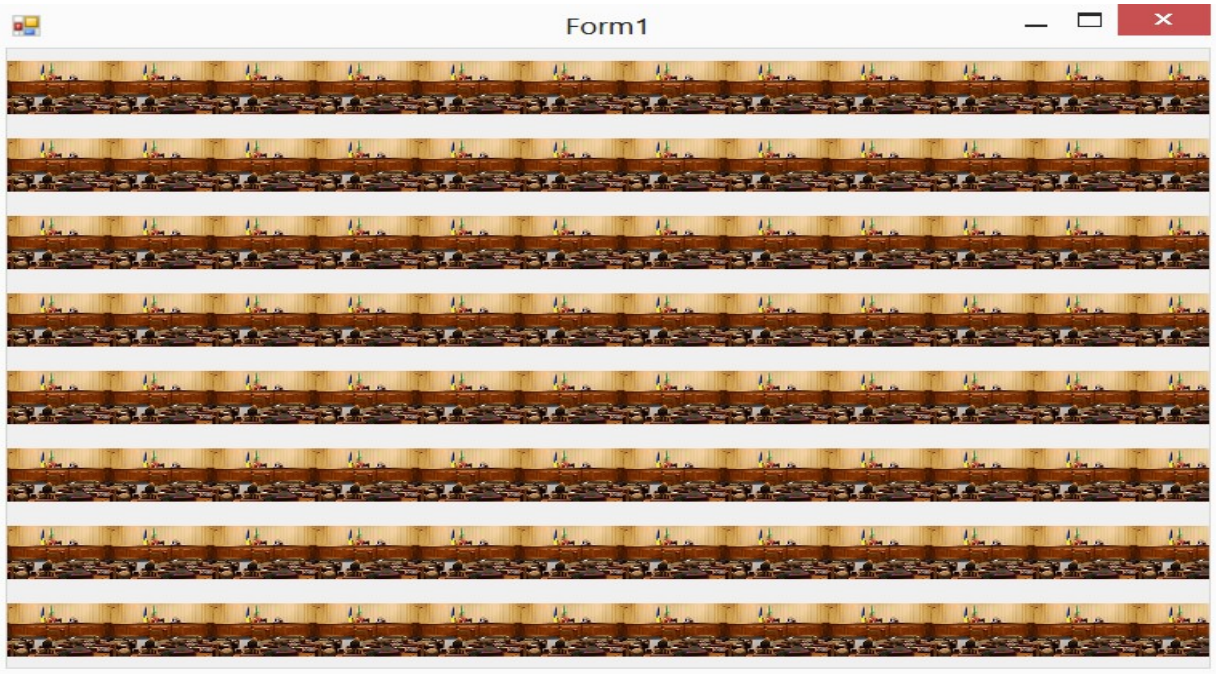


Рис. 5.3. До зміни розмірів зображення

5.4. Клас Bitmap

Ми обговорювали роботу з об'єктами типу *Image*. Простір імен *System-Drawing* включає клас *Bitmap*, успадкований від Image. Всі властивості *Image* застосовні і до *Bitmap*. Будь-які операції, допустимі щодо *Image*, допустимі й щодо *Bitmap*. Клас *Bitmap* дозволяє працювати прямо з бітами карти. У класі *Image* немає конструкторів, а у *Bitmap* їх 12. Наступні конструктори завантажують об'єкт *Bitmap* з файлу, потоку чи ресурсу.

```
Bitmap(string strFilename);
Bitmap(string strFilename, bool bUselmageColorManagement);
Bitmap(Stream stream);
Bitmap(Stream stream, bool bUselmageColorManagement);
Bitmap(Type type, string strResource).
```

Перші чотири конструктори дублюють статичні методи *FromFile* та *FromStream* класу *Image*. П'ятий конструктор завантажує об'єкт *Bitmap* з ресурсів, які зазвичай вбудовуються в *.exe* - файлах додатків. Далі наведено набір конструкторів, котрі створюють нові об'єкти *Bitmap* на основі існуючих об'єктів *Image*.

```
Bitmap(Image image)
Bitmap(Image image, Size size)
Bitmap(Image image, int cx, int cy)
```

Хоча перший аргумент цих конструкторів визначений як *Image*, він може бути іншим об'єктом *Bitmap*. Перший конструктор аналогічний методу *Clone* класу *Image* – він створює ідентичний вихідному новий об'єкт *Bitmap*. Другий і третій схожі на метод *GetThumbnailImage* - вони змінюють розмір зображення, У всіх випадках нова бітова карта успадковує формат пікселів вихідної карти, а роздільна здатність нової карти встановлюється рівною роздільній здатності дисплея. Останні чотири конструктори не мають аналогів у класі *Image*. Вони створюють нові об'єкти *Bitmap* з порожніми зображеннями:

```
Bitmap(int cx, int cy)
Bitmap(int cx, int cy, PixelFormat pf)
Bitmap(int cx, int cy, Graphics grfx)
Bitmap(int cx, int cy, int cxRowBytes, PixelFormat pf, IntPtr pBytes)
```

Перші три ініціалізують пікселі значенням 0, яке у різних форматах бітових карт має різний сенс. У бітових *RGB-картах* 0 відповідає чорному кольору. В *ARGB-картах* 0 означає прозорість. Четвертий також може приймати вказівник на масив байтів, що ініціалізує растрове зображення.

Перший конструктор створює об'єкт *Bitmap* заданого розміру з форматом пікселів *PixelFormatormat32bppArgb*. Горизонтальна і вертикальна роздільна здатність задаються рівними роздільною здатністю дисплея.

Другий конструктор дозволяє вказати елемент перерахування *PixelFormat*, якщо не влаштовує формат *PixelFormatformat32bppArgb*.

Третій конструктор дозволяє вказати об'єкт *Graphics*. Незалежно від того, пов'язаний зазначений об'єкт з дисплеєм або принтером, і незалежно від того, чи кольоровий принтер. Цей конструктор завжди створює об'єкт *Bitmap* з форматом пікселів *PixelFormatformat32bppArgb*.

Якщо створюються бітові карти, роздільна здатність яких не відповідає ні екранній роздільній здатності, ні роздільній здатності принтера, можна змінити роздільну здатність завантаженої або створеної бітової карти за допомогою методу *SetResolution* класу *Bitmap*:

```
void SetResolution(float xDpi, float yDpi);
```

Для малювання на поверхні бітової карти необхідно створити об'єкт *Graphics* для даної карти та малювати на ній, як на звичайному графічному пристрої.

Приклад 69.

У програмі на лістингу 5.4 створюється бітова карта і на ній виводиться текст висотою у 72 пункти. Потім бітова карта виводиться в клієнтській області та (не обов'язково) на сторінці принтера. Результат наведено на рис. 5.4.

Лістинг 5.4. (examp69)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp69
{
   public partial class Form1 : Form
   {
      const float fResol = 300;
      Bitmap bitmap;

      public Form1()
      {
         InitializeComponent();
         ResizeRedraw = true;
         Size = new Size(600, 300);
         Text = "Hello, World!";
         bitmap = new Bitmap(3,3);
            //bitmap.SetResolution(fResol,fResol);
         Graphics gr = Graphics.FromImage(bitmap);
         Font font = new Font("Times New Roman", 72);
         Size size = gr.MeasureString(Text, font).ToSize();
         bitmap = new Bitmap(bitmap,size);
            //bitmap.SetResolution(fResol,fResol);
         gr.Dispose();
         gr = Graphics.FromImage(bitmap);
         gr.Clear(Color.White);
         gr.DrawString(Text, font, Brushes.Black, 0, 0);
         gr.Dispose();
```

```
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            base.OnPaint(e);
            e.Graphics.DrawImage(bitmap, 100, 0);
        }
    }
}
```

Рис. 5.4. Створення бітової карти

5.5. Закріплення іконки до форми

Доволі часто корисно зберігати невеликі двійкові файли (зокрема, бітові карти, значки та нестандартні курсори) прямо в *.exe-файлі* програми. Таким чином вони не зможуть загубитися. Файли, що зберігаються у файлі, називаються ресурсами. Найпростіший спосіб прикріплення іконки як ресурсу до форми виконується встановленням властивості *icon*. У цій властивості можна задати іконку для форми. Тут також є кнопка, яка викликає стандартне вікно для відкриття файлу. Завантажений значок потрапить у *resx-файл* форми. Крім того, у редакторі ресурсів можна створити файл **.ico*. Окрім *resx-файлу*, можна підключити іконку в конструкторі форми. Це показано у трьох варіантах на лістингу 5.5.

Приклад 70.

Лістинг 5.5. (examp70)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace exam50
{
    public partial class Form1: Form
    {
        public Form1()
        {
            InitializeComponent();
//      Вариант 1
                                //создадим объект иконки и свяжем его с формой
                                //в конструктор необходимо передать строку с именем файла иконки
                                // Icon icon = new Icon("icon_name.ico");
                                // this.Icon = icon;
//      Вариант 2
```

```
//    Icon = exam52.Properties.Resources.Icon1;

//    Вариант 3

        // Icon = (Icon)exam52.Properties.Resources.ResourceManager.GetObject("Icon1");
    }
  }
}
```

На рис 5.5. показано іконку. У прикладі examp70 використовується перший варіант програмного вставлення іконки.

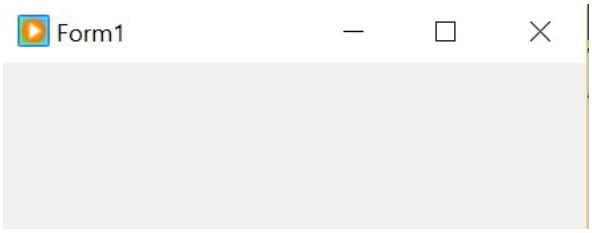


Рис. 5.5. Вставлення іконки.

6. Типові додатки

6.1. Приклад гри в друкарську машинку

Приклад 71.

Нехай у формі випадково з'являються літери через інтервал часу. Ці літери відображаються у *ListBox*. Паралельно з цим гравець вводить букви з консолі. Якщо введена користувачем буква є на екрані (*ListBox*) підвищується оцінка гри. По мірі введення літер гра ускладнюється. Як тільки форма (*ListBox*) заповнена заданою кількістю літер, гра закінчується. На рис. 6.1. показані літери, що згенеровані випадковим чином.

За допомогою елемента керування *StatusStrip* створюється статусний рядок, що включає результати гри. У статусний рядок додано 5 елементів *StatusLabel* та елемент *StatusBar*:

Correct - кількість правильно введених літер (літер, які містяться в ListBox).
Missed - кількість пропущених літер.
Total - кількість введених літер.
Accuracy - відсоток правильно введених літер.
Difficulty - мітка для StatusBar, яка відображає відсоток правильно введених літер.

Рис. 6.1. Літери в ListBox

Текст програми наведено у лістингу 6.1.

Лістиг 6.1. (examp71)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp71
{
   public partial class Form1 : Form
   {
      Random random = new Random();
      Stats stats = new Stats();

      public Form1()
      {
         InitializeComponent();
      }
      private void timer1_Tick(object sender, EventArgs e)
      {
         listBox1.Items.Add((Keys)random.Next(65,90));
         if (listBox1.Items.Count > 15)
         {
            listBox1.Items.Clear();
            listBox1.Items.Add(" Игра закончена");
            timer1.Stop();
         }
      }

      private void Form1_KeyDown(object sender, KeyEventArgs e)
      {
         if (listBox1.Items.Contains(e.KeyCode))
         {
            listBox1.Items.Remove(e.KeyCode);
            listBox1.Refresh();
            if (timer1.Interval > 400) timer1.Interval -= 10;
            if (timer1.Interval > 250) timer1.Interval -= 7;
            if (timer1.Interval > 100) timer1.Interval -= 2;
            difficultyProgressBar.Value = 800 - timer1.Interval;
            stats.Update(true);
         }
         else  stats.Update(false);
         correctLabel.Text  = "Correct:" + stats.Correct;
         missedLabel.Text   = "Missed:"   + stats.Missed;
         totalLabel.Text    = "Total:"     + stats.Total;
         accuracyLabel.Text = "Accuracy:" + stats.Accuracy;
      }
    }

   public class Stats
   {
      public int Total = 0;
      public int Missed = 0;
      public int Correct = 0;
      public int Accuracy = 0;
      public void Update(bool correctKey)
      {
```

```
        Total++;
        if (!correctKey) Missed++;
        else Correct++;
        Accuracy = 100 * Correct / (Missed + Correct);
      }
    }
  }
```

Немає сенсу докладно описувати текст програми. Студент самостійно повинен зрозуміти програму, тим більше що програму (examp71) було повністю наведено.

6.2. Робота з елементами керування як об'єктами

Приклад 72.

У прикладі розглядається переміщення по горизонталі трьох міток. Їхнє переміщення та зупинка виконується при натисканні відповідних трьох кнопок. Текст програми та фрагмент виконання наведено відповідно у лістингу 6.2. та рис. 6.2.

Лістинг 6.2. (examp72)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp72
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

            LabelBouncer[] bouncers =
            new LabelBouncer[3];

        private void ToggleBouncing(int index, Label LabelToBounce)
        {
            if (bouncers[index] == null)
            {
                bouncers[index] = new LabelBouncer();
                bouncers[index].MyLabel = LabelToBounce;
            }


            else bouncers[index] = null;
        }
```

```
        private void timer1_Tick(object sender, EventArgs e)
        {
            for (int i = 0; i < 3; i++)
            if (bouncers[i] != null) bouncers[i].Move();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            ToggleBouncing(0,label1);
        }

        private void button2_Click(object sender, EventArgs e)
        {
            ToggleBouncing(1, label2);
        }
        private void button3_Click(object sender, EventArgs e)
        {
            ToggleBouncing(2, label3);
        }
    }

    class LabelBouncer
    {
        public Label MyLabel;
        public bool GoingForward = true;
        public void Move()
        {
            if (MyLabel == null)return;
            if (GoingForward == true)
            {
                MyLabel.Left += 5;
                if (MyLabel.Left >= MyLabel.Parent.Width
                - MyLabel.Width) GoingForward = false;
                return;
            }
                MyLabel.Left -= 5;
                if (MyLabel.Left <= 0) GoingForward = true;
         }// Move
     }// class LabelBouncer
}
```

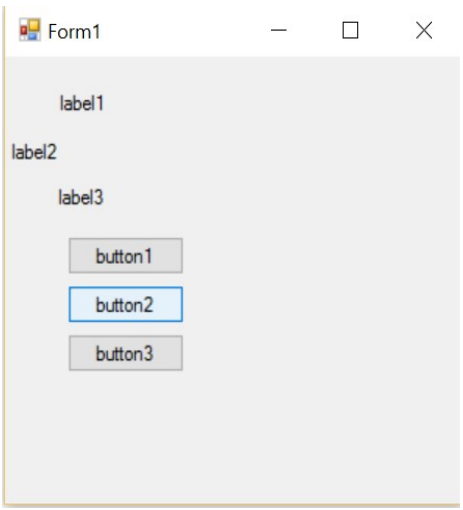


Рис. 6.2. Керування переміщенням кнопок.

6.3. Формування меню для кафе

Пример 73.

Для кафе формується обід, що складається з першого, другого та третього. При цьому випадково формуються складові меню. У формі 6 міток, які виводять 6 запропонованих меню. На рис. 6.3. показано результат виконання програми. Текст програми наведено у лістингу 6.3.

Лістинг 6.3. (examp73)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp73
{
   public partial class Form1 : Form
   {
      MenuMaker menu;
      public Form1()
      {
         InitializeComponent();
         menu = new MenuMaker();
         menu.Randomizer = new Random();
         Text = "М Е Н Ю";
      }

      private void button1_Click(object sender, EventArgs e)
      {
         label1.Text = menu.GetMenuItem();
         label2.Text = menu.GetMenuItem();
         label3.Text = menu.GetMenuItem();
         label4.Text = menu.GetMenuItem();
         label5.Text = menu.GetMenuItem();
         label6.Text = menu.GetMenuItem();
      }

      class MenuMaker
      {
         public Random Randomizer;

         string[] Первые  = {"Суп  Харчо      ", "Борщ Украин.",
                             "Рассольник   ", "Бульон Курин.", "Солянка          " };
         string[] Вторые  = {"Говядина", "Курица    ",
                             "Баранина", "Перепела", "Свинина  " };
         string[] Напитки = {"Сок вишневый", "Сок персиковый",
                             "Компот (С/Ф)", "Кисель (слива)", "Кефир" };

         public string GetMenuItem()
         {
            string per =
            Первые[Randomizer.Next(Первые.Length)];
            string vtor =
            Вторые[Randomizer.Next(Вторые.Length)];
            string tret =
            Напитки[Randomizer.Next(Напитки.Length)];
```

```
            return per + "   " + vtor
            + "    " + tret;
        }
    }//class MenuMaker
} // form1
}// namespace
```

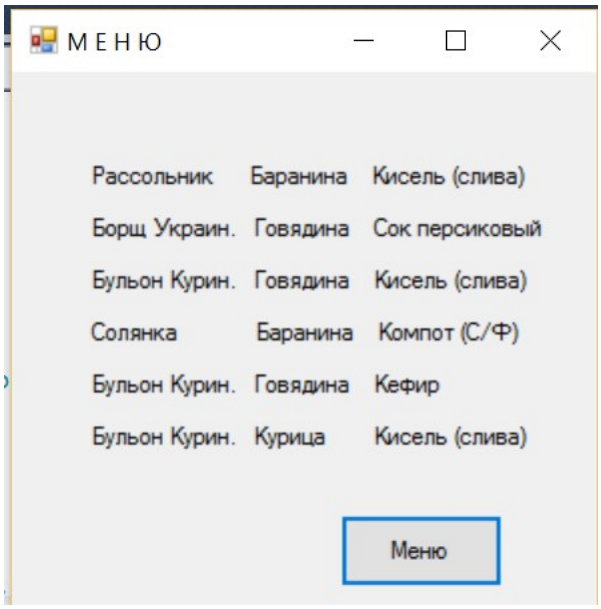


Рис. 6.3. До програмування меню

6.4. Оцінювання вартості обіду

Вартість обіду в ресторані визначається так.

За кожного гостя – $25.

1. Більшість обідів сервірується з алкогольними напоями - $20 на особу
2. Можна вибрати "здоровий варіант" - це коштує всього $5 на людину, замість алкоголю подаються соки та вода. "Здоровий" варіант набагато простіше в організації, тому знижка становитиме 5% на всю вартість заходу.
3. Звичайне оформлення коштує $7.5 на особу і плюс початковий внесок $30. Вартість особливого оформлення збільшується до $15 на особу, а початковий внесок становитиме $50.

Приклад 74.

На рис. 6.4. показано результат виконання програми. У лістингу 6.4. наведено прогаму реалізації завдання. При цьому крім основного класу форми *Form1* використовується додатковий клас *Rs*. Вони описані в різних файлах і мають спільний *namespace*. У файлі *Rs* виконуються розрахунки. В основній формі (клас *Form1*) виконується інтерфейс із користувачем, виконується формування та обробка відгуків, у функціях яких виконуються розрахунки методами класу *Rs*. Результати розрахунків виводяться у *TextBox* основної форми.

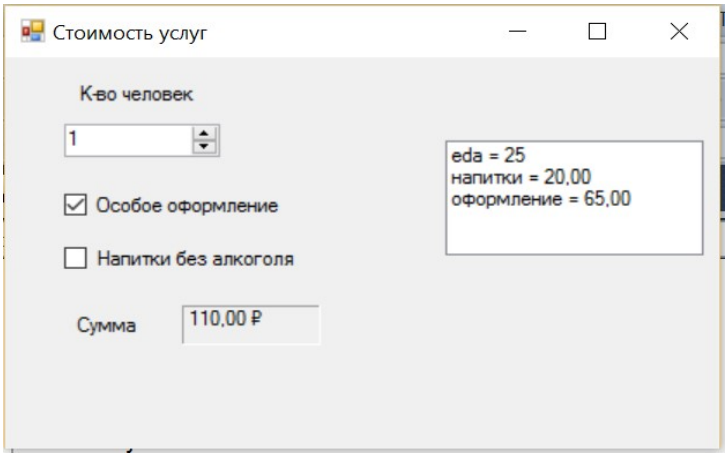


Рис. 6.4. До оцінювання вартості обіду

Лістинг 6.4. (examp74)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace ot74
{
    public partial class Form1 : Form
    {
        Rs rs;

        public Form1()
        {
            InitializeComponent();
            rs = new Rs();
            rs.Set_nap (checkBox1.Checked);
            rs.Set_oform (checkBox2.Checked);
            rs.kol_chel = (int)numericUpDown1.Value;
            Display();
        }

        private void checkBox1_CheckedChanged(object sender, EventArgs e)// напитки
        {
            rs.Set_nap(checkBox1.Checked);
            Display();
        }

        private void checkBox2_CheckedChanged(object sender, EventArgs e) //
оформление
        {
            rs.Set_oform(checkBox2.Checked);
            Display();
        }

        private void numericUpDown1_ValueChanged(object sender, EventArgs e)
        {
            rs.kol_chel = (int)numericUpDown1.Value;
            rs.Set_oform(checkBox2.Checked);
            Display();
        }
```

```
        private void Display()
        {
            decimal Cost = rs.Stoim(checkBox1.Checked);
            listBox1.Items.Clear();
            listBox1.Items.Add("eda = " + rs.St_ed * rs.kol_chel);
            listBox1.Items.Add("напитки = " +  rs.St_napit * rs.kol_chel);
            listBox1.Items.Add("оформление = " + rs.St_oform);
            decimal sk = ((25M + rs.St_napit) * rs.kol_chel + rs.St_oform) * 0.05M;
            if (checkBox1.Checked) listBox1.Items.Add("скидка ( 5% ) = " + sk);
            label3.Text = Cost.ToString("c");
        }
    }
}
```

Програма у файлі Rs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ot74
{
    class Rs
    {
        public int St_ed = 25;
        public int kol_chel;
        public decimal St_napit;
        public decimal St_oform;

        public void Set_nap(bool vr)
        {
            St_napit = vr ? 05.00M : 20.00M;
        }

        public void Set_oform(bool vr)
        {
            St_oform = vr ? kol_chel * 15.00M + 50.00M : kol_chel * 7.50M + 30.00M;
        }

        public decimal  Stoim (bool vr)
        {
            decimal Total_st;
            Total_st = St_oform + (St_napit + St_ed) * kol_chel;
            if (vr) Total_st*= 0.95M;
            return Total_st;
        }
    }
}
```

6.5. Постановка задачі вартості обіду та днів народження

У розділі 6.4 розроблено програму вартості обіду. У цьому розділі цю програму розширено. Додатково виконується оцінка днів народження. Вибір одного з цих варіантів (розв’язуваних задач) виконується елементом керування *TabControl*. Елементи керування для першої та другої задач відповідно встановлюються на першій та другій сторінці *TabControl*

(*tabPage1* та *tabPage2*). Під час виконання завдань їх перемикання виконується перемиканням клавіш *TabControl*.

Постановка другої задачі оцінки вартості днів народження.

1. За кожного гостя $25.
2. Звичайне оформлення коштує $7.5 на особу плюс початковий внесок $30. Вартість особливого оформлення збільшується до $15 на особу, а початковий внесок – $50.
3. Якщо учасників менше чотирьох, готуємо 8-дюймовий торт ($40). Якщо учасників більше чотирьох готуємо 16-дюймовий торт($75)
4. Напис на торті $0.25 за букву. На першому торті міститься до 16 літер, на другому – до 40.

Для реалізації завдання розглядається три варіанти:

Мистецтво програмування залежить від вибору структури програмування. Від цього залежить надійність реалізації завдання, можливість розширення та коригування програмного забезпечення. Розглядається три варіанти реалізації програмного забезпечення. Слід зазначити, що умова завдання взята у книзі Стілльєна [1]. Проте реалізацію програмного забезпечення змінено та значно розширено. Для підвищення рівня програмування студентів не надається детальне пояснення варіантів. Це більше відноситься до другого та третього варіанта програмування. Розібравши ці варіанти, студент самостійно зможе обрати зручну для нього структуру програмування. Вивчивши варіанти, програміст може вибрати один з них або на основі вивчення прикладів, що розглядаються, запропонувати більш ефективний. Це залежить передусім від розглянутого завдання.

6.6. Оцінювання вартості обіду та днів народження (Варіант 1)

Приклад 75.

Програмне забезпечення складається з 4 файлів, кожен із яких включає один клас.

Основний клас – *Form1*, який виконує розрахунки методами розрахункових класів.

Клас *obed* використовується для розрахунку обідів

Клас *D_R* використовується для розрахунку днів народження.

Клас *Basa_kl* розроблено на основі наступного. Обидва завдання мають однакові розрахунки (вартість їжі на одну особу, вартість оформлення). Тому клас, який розглядається, є базовим щодо успадкованих *obed*, *D_R*.

Слід звернути увагу до програмування обмеження написів на торті. Немає необхідності детально описувати програмне забезпечення. Ця можливість мала місце при вивченні попереднього матеріалу цього посібника.

Текст програми наведено в лістингу 6.5. Результати розв’язання задач наведено на рис. 6.5. та 6.6.

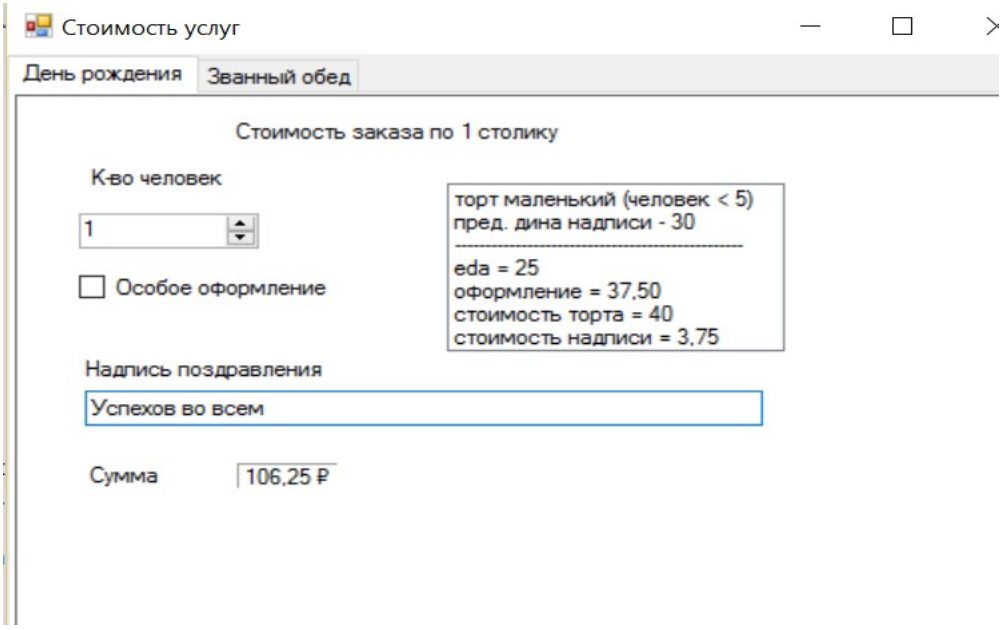


Рис. 6.5. Оцінювання вартості днів народження

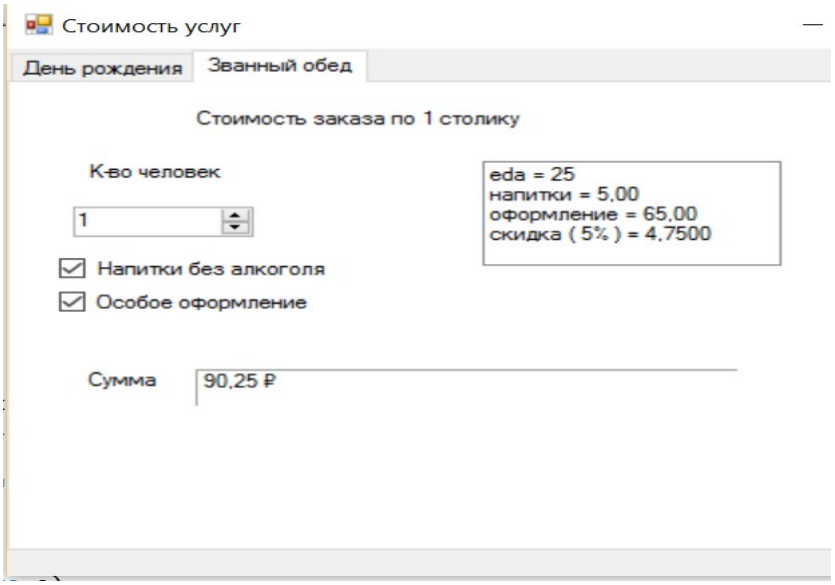


Рис. 6.6. Оцінювання вартості званого обіду

Лістинг 6.5. (examp75)

class Form1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp75
{
    public partial class Form1 : Form
```

```
    {
        bool klav_Back = false;
        Obed rs;
        D_R  dr;

        public Form1()
        {
            InitializeComponent();

            rs = new Obed();
            dr = new D_R();

            rs.kol_chel = (int)numericUpDown1.Value;
            dr.kol_chel = (int)numericUpDown2.Value;

            rs.Set_nap   (checkBox1.Checked);
            rs.Set_oform (checkBox2.Checked);

            dr.Set_oform (checkBox3.Checked);
            textBox1.Focus();
            Display();
        }

        private void checkBox1_CheckedChanged(object sender, EventArgs e)// напитки
        {
            rs.Set_nap(checkBox1.Checked);
            Display();
        }

        private void checkBox_CheckedChanged(object sender, EventArgs e) //
оформление
        {
            rs.Set_oform(checkBox2.Checked);
            dr.Set_oform(checkBox3.Checked);
            Display();
        }

        private void numericUpDown_ValueChanged(object sender, EventArgs e)
        {
            rs.kol_chel = (int)numericUpDown1.Value;
            dr.kol_chel = (int)numericUpDown2.Value;
            rs.Set_oform(checkBox2.Checked);
            dr.Set_oform(checkBox3.Checked);
            Display();
        }
        // ------------------------------------------------------------------------
        private void Display()
        {
            label10.Visible = false;
            decimal Cost;
            dr.kol_sim = textBox1.Text.Length;
            //this.Text = dr.kol_sim.ToString();
            Cost = rs.Stoim(checkBox1.Checked);
            label3.Text = Cost.ToString("c");
            Cost = dr.Stoim(checkBox3.Checked);
            label8.Text = Cost.ToString("c");

            listBox1.Items.Clear();
            listBox1.Items.Add(rs.mas_str[0]);
            listBox1.Items.Add(rs.mas_str[1]);
            listBox1.Items.Add(rs.mas_str[2]);
            listBox1.Items.Add(rs.mas_str[3]);
```

```
            listBox2.Items.Clear();
            listBox2.Items.Add(dr.mas_str[0]);
            listBox2.Items.Add(dr.mas_str[1]);
            listBox2.Items.Add(dr.mas_str[2]);
            listBox2.Items.Add(dr.mas_str[3]);
            listBox2.Items.Add(dr.mas_str[4]);
            listBox2.Items.Add(dr.mas_str[5]);
            listBox2.Items.Add(dr.mas_str[6]);

            if (dr.kol_chel < 5 && textBox1.Text.Length > dr.max_dl_m)
            textBox1.Text = textBox1.Text.Substring(0, dr.max_dl_m);

            if (textBox1.Text.Length >= (dr.kol_chel < 5 ? dr.max_dl_m :
dr.max_dl_b))
            label10.Visible = true;
        }

        private void textBox1_KeyDown(object sender, KeyEventArgs e)
        {
            klav_Back = false;
            if (e.KeyCode == Keys.Back) klav_Back = true;
        }

        private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
        {
                if (textBox1.Text.Length >= (dr.kol_chel < 5 ? dr.max_dl_m :
dr.max_dl_b)
                && klav_Back == false)e.Handled = true;
        }

        private void textBox1_KeyUp(object sender, KeyEventArgs e)
        {
            Display();
        }
   }
}
.....................................................................................
....
```

class Basa_kl

Лістинг 6.6.

```
sing System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace examp75
{
    class Basa_kl
    {

        public int       kol_chel;
        private  decimal  St_ed = 25;
        protected  decimal St_oform;
        //public int kol_sim ;

        public void Set_oform(bool vr)
        {
            St_oform = vr ? kol_chel * 15.00M + 50.00M : kol_chel * 7.50M + 30.00M;
        }
```

```
        virtual public decimal Stoim (bool vr)
        {

            return (St_oform + St_ed * kol_chel);
        }
    }
}
```

.....

class Obed

Лістинг 6.7.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace examp75
{
    class Obed : Basa_kl
    {
        private decimal St_napit;
        public string[] mas_str = new string[5];

        public void Set_nap (bool vr)
        {
            St_napit =   vr ? 05.00M : 20.00M;
        }

         override public decimal Stoim(bool vr)
        {
            decimal Total_st = base.Stoim(vr) + St_napit * kol_chel;

             mas_str[0] = "eda = " + 25M * kol_chel;
             mas_str[1] = "напитки = " + St_napit * kol_chel;
             mas_str[2] = "оформление = " + St_oform;
             mas_str[3] = "";
             if(vr) mas_str[3] = "скидка ( 5% ) = " + Total_st * 0.05M;
             return vr ? Total_st * 0.95M :Total_st;
         }
    }
}
```

.....

.

class D_R

Лістинг 6.8.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

   namespace examp75
   {
       class D_R : Basa_kl
        {
                   public int kol_sim;
                   public string[] mas_str = new string[7];
                   public int max_dl_m = 30;
                   public int max_dl_b = 50;
               private decimal St_tor()
               {
                   return kol_chel < 5 ? 40 : 80;
```

```
            }
                override public decimal Stoim(bool vr)
            {
                mas_str[0] = "торт " +
                (kol_chel < 5 ? "маленький (человек < 5)" : " большой (человек >
4)");
                mas_str[1] = "пред. дина надписи - " +
                (kol_chel < 5 ? max_dl_m : max_dl_b);
                mas_str[2] = "-----------------------------------------------";
                mas_str[3] = "eda = " + 25M * kol_chel;
                mas_str[4] = "оформление = " + St_oform;
                mas_str[5] = "стоимость торта = " + St_tor();
                mas_str[6] = "стоимость надписи = " +
                kol_sim * 0.25M;
                return ( base.Stoim(vr) +
                St_tor() + kol_sim * 0.25M );
            }
        }
    }
```

6.7. Оцінювання вартості обіду та днів народження (Варіант 2)

Приклад 76.

В даному варіанті використовується основний клас *Form1*. Він використовує два діалоги, класи яких похідні від *Form*. Цими класами є *obed* та *D_R*.

У першому діалозі *obed* встановлюються елементи керування формування вихідних даних і виконання окремих розрахунків обіду. Ці дані передаються до основного файлу *Form1*. Тут розгорнута інформація щодо розрахунку обіду детально подається в *ListBox* основного класу. Аналогічним чином обчислюється вартість дня народження в діалозі *D_R*.

У *ListBox* дається інформація про витрати за днями народження. На рис. 6.7. та рис. 6.8. показано фрагменти розв'язання завдань відповідно при розрахунку обіду та днів народження. Програмування класів *Form1*, *obed* та *D_R* наведено відповідно у лістингах 6.9, 6.10,. 6.11.

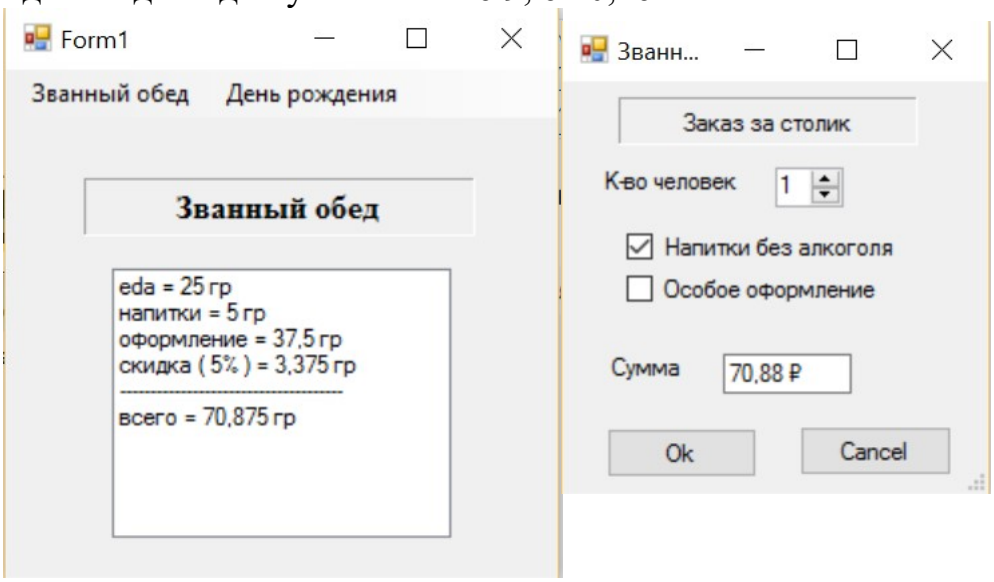


Рис. 6.7. До розрахунку обіду

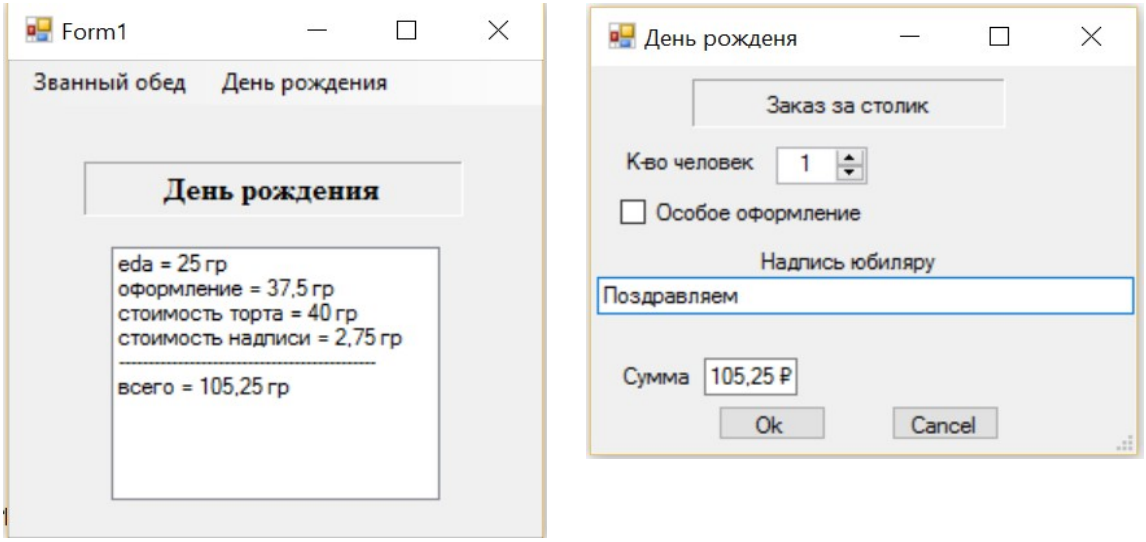


Рис. 6.8. До розрахунку вартості днів народження

class Form1

Лістинг 6.9. (examp76)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp76
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            label1.Visible = false;
            listBox1.Visible = false;
        }
        private void обедToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Obed dlg = new Obed();
            dlg.ShowDialog();
            label1.Visible = false;
            listBox1.Visible = false;
            listBox1.Items.Clear();
            if (dlg.DialogResult == DialogResult.OK)
            {
                label1.Visible = true;
                listBox1.Visible = true;
                label1.Text = "Званный обед";
                listBox1.Items.Clear();
                listBox1.Items.Add("eda = " + dlg.mas[0] + " гр");
                listBox1.Items.Add("напитки = " + dlg.mas[1] + " гр");
                listBox1.Items.Add("оформление = " + dlg.mas[2] + " гр");
                listBox1.Items.Add((dlg.mas[3] > 0.00001 ?
                "скидка ( 5% ) = " : "скидка  = ") + dlg.mas[3] + " гр");
                listBox1.Items.Add("-----------------------------------");
                listBox1.Items.Add("всего = " + (dlg.mas[0] + dlg.mas[1]
+ dlg.mas[2] + dlg.mas[3]) + " гр"); } }
```

```
        private void деньРожденияToolStripMenuItem_Click(object sender, EventArgs e)
        {
            D_R dlg = new D_R();
            dlg.ShowDialog();
            label1.Visible = false;
            listBox1.Visible = false;
            listBox1.Items.Clear();

            if (dlg.DialogResult == DialogResult.OK)
            {
                label1.Visible = true;
                listBox1.Visible = true;
                label1.Text = "День рождения";
                listBox1.Items.Clear();
                listBox1.Items.Add("eda = " + dlg.mas[0] + " гр");
                listBox1.Items.Add("оформление = " + dlg.mas[1] + " гр");
                listBox1.Items.Add("стоимость торта = " + dlg.mas[2] + " гр");
                listBox1.Items.Add("стоимость надписи = " + dlg.mas[3] + " гр");
                listBox1.Items.Add("----------------------------------------");
                listBox1.Items.Add("всего = " +
                (dlg.mas[0] + dlg.mas[1] + dlg.mas[2] + dlg.mas[3])
                + " гр");
            }
        }
    }
}
```

.....

class obed

Лістинг 6.10.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp76
{
    public partial class Obed : Form
    {

        private int kol_chel;
        public  double[] mas = new double[4];

        public Obed()
        {
            InitializeComponent();
            Display();

        }
        private void button1_Click(object sender, EventArgs e)
        {
            DialogResult = DialogResult.OK;
        }
        private void button2_Click(object sender, EventArgs e)
        {
            DialogResult = DialogResult.Cancel;
        }
```

```
        private void otkl(object sender, EventArgs e)
        {
            Display();
        }
        void Display()
        {
          kol_chel = (int)numericUpDown1.Value;
          mas[0] = 25.0 * kol_chel;
          mas[1] =  kol_chel * (checkBox1.Checked ? 5.0 : 20.0); // напитки
          mas[2] =  checkBox2.Checked  ?
          kol_chel * 15.00 + 50.00 : kol_chel * 7.50 + 30.00;   // оформление
          mas[3] = 0;                                           // скидки
          if(checkBox1.Checked) mas[3] =
          (mas[0] + mas[1] + mas[2])*0.05;
          textBox1.Text =
          (mas[0] + mas[1] + mas[2] + mas[3]).ToString("c");
        }
    }
}
```

.....

class D_R

Лістинг 6.11.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp76
{
    public partial class D_R : Form
    {
        private int kol_chel;
        private int kol_sim;
        public double[] mas = new double[4];
        bool klav_Back = false;
        public int max_dl_m = 30;
        public int max_dl_b = 50;
        public D_R()
        {
            InitializeComponent();
            Display();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            DialogResult = DialogResult.OK;
        }
        private void button2_Click(object sender, EventArgs e)
        {
            DialogResult = DialogResult.Cancel;
        }
        private void otkl(object sender, EventArgs e)
        { Display(); }
        private void textBox1_KeyDown(object sender, KeyEventArgs e)
        {
            klav_Back = false;
            if (e.KeyCode == Keys.Back) klav_Back = true;
        }
```

```
        private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (textBox1.Text.Length >= (kol_chel < 5 ? max_dl_m : max_dl_b)
            && klav_Back == false) e.Handled = true;
        }
        private void textBox1_KeyUp(object sender, KeyEventArgs e)
        { Display(); }
        void Display()
        {
            label5.Visible = false;
            kol_chel = (int)numericUpDown1.Value;
            kol_sim = textBox1.Text.Length;
            mas[0] = 25.0 * kol_chel;
            mas[1] = checkBox1.Checked ?
            kol_chel * 15.00 + 50.00 : kol_chel * 7.50 + 30.00;   // оформление
            mas[2] = kol_chel < 5 ? 40.0 : 80.0; // стоимость торта
            mas[3] = kol_sim * 0.25;
            textBox2.Text =
            (mas[0] + mas[1] + mas[2] + mas[3]).ToString("c");
            if (kol_chel < 5 && textBox1.Text.Length > max_dl_m)
            textBox1.Text = textBox1.Text.Substring(0, max_dl_m);
            if (textBox1.Text.Length >= (kol_chel < 5 ? max_dl_m : max_dl_b))
            label5.Visible = true;
        }
    }
}
```

6.8. Оцінювання вартості обіду та днів народження (Варіант 3)

Приклад 77.

У третьому варіанті використовується лише один клас *Form1*. Результат виконання програми та текст програми класу показані відповідно на рис. 6.9 та лістингу 6.12.

Цей варіант вимагає використовувати однин файл, проте через це ускладнюється програмний код. Для складних завдань, що включають модулі з різними функціональними призначеннями, використання такого варіанту складне. Як зазначалося, студент самостійно зможе обрати зручну йому структуру програмування чи запропонувати свою ефективнішу.

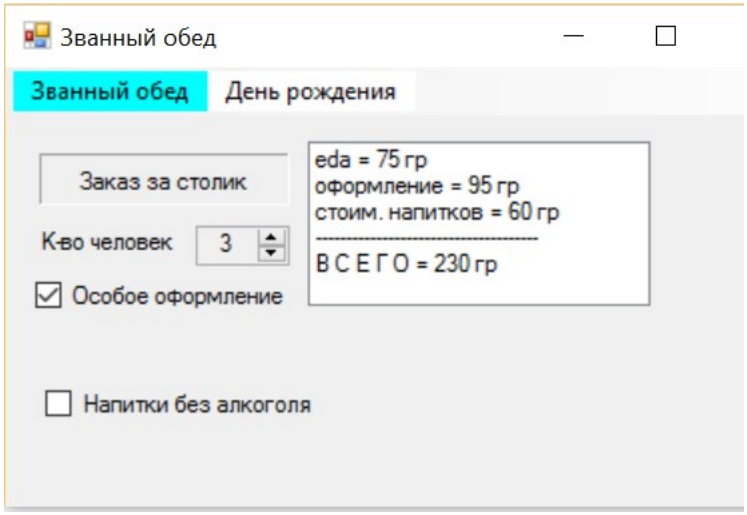


Рис. 6.9. До розрахунку вартості званого обіду та днів народження

Лістинг 6.12. (examp77)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace examp77
{
    public partial class Form1 : Form
    {
        bool obed = true;
        double[] mas = new double[5];
        int  kol_chel, kol_sim;
        bool klav_Back = false;
        int max_dl_m = 30;
        int max_dl_b = 50;

        public Form1()
        {
            InitializeComponent();
            this.checkBox2.Location = new System.Drawing.Point(20, 160);
            this.деньРожденияToolStripMenuItem.BackColor = Color.White;
            this.званныйОбедToolStripMenuItem.BackColor = Color.Cyan;
            ekran(obed);
            Display();
        }

        private void званныйОбедToolStripMenuItem_Click(object sender, EventArgs e)
        {
           if(obed) return;
           ekran(obed = true);
           this.деньРожденияToolStripMenuItem.BackColor = Color.White;
           this.званныйОбедToolStripMenuItem.BackColor = Color.Cyan;
           Display();
        }

        private void деньРожденияToolStripMenuItem_Click_1(object sender, EventArgs
e)
        {
            if (!obed) return;
            ekran(obed = false);
            this.деньРожденияToolStripMenuItem.BackColor = Color.Cyan;
            this.званныйОбедToolStripMenuItem.BackColor = Color.White;
            Display();
        }

        private void ekran(bool obed)
        {

            this.checkBox2.Visible = obed;
            this.label3.Visible = this.label5.Visible =
            this.textBox1.Visible = !obed;
            Text = (obed) ? "Званный обед" : "День рождения";
        }

        private void Display()
        {
           label5.Visible = false;
```

```
            this.textBox1.Focus();
            kol_chel = (int)numericUpDown1.Value;
            if (kol_chel == 0) return;
             mas[0] = 25.0 * kol_chel;
            mas[1] = checkBox1.Checked ?
            kol_chel * 15.00 + 50.00 : kol_chel * 7.50 + 30.00;   // оформление
            if (obed)
            {
                mas[2] = kol_chel * (checkBox2.Checked ? 5.0 : 20.0); // напитки
                mas[3] = checkBox2.Checked ? (mas[0] + mas[1] + mas[2]) * -0.05 : 0;//
скидки
            }
            else
            {
                mas[2] = kol_chel < 5 ? 40.0 : 80.0; // стоимость торта
                if (kol_chel < 5 && textBox1.Text.Length > max_dl_m)
                textBox1.Text = textBox1.Text.Substring(0, max_dl_m);
                kol_sim = textBox1.Text.Length;
                mas[3] = kol_sim * 0.25;// стоимость надписи
            }
             listBox1.Items.Clear();
             listBox1.Items.Add("eda = " + mas[0] + " гр");
             listBox1.Items.Add("оформление = " + mas[1] + " гр");
             listBox1.Items.Add((obed ? "стоим. напитков = " :
             "стоим. торта = ") + mas[2] + " гр");
             if(Math.Abs(mas[3]) > 0.00001)
             listBox1.Items.Add((obed ?"скидка ( 5% ) = " :
             "стоим. надписи = ") + mas[3] + " гр");
             listBox1.Items.Add("-------------------------------------");
             listBox1.Items.Add("В С Е Г О = " +
             (mas[0] + mas[1] + mas[2] + mas[3]) + " гр");
        }

        private void numericUpDown1_ValueChanged(object sender, EventArgs e)
        {
            Display();
        }

        private void checkBox_Checked(object sender, EventArgs e)
        {
            if ((CheckBox)sender == this.checkBox1
            || (CheckBox)sender == this.checkBox2) Display();
        }

        private void textBox1_KeyDown(object sender, KeyEventArgs e)
        {
            klav_Back = false;
            if (e.KeyCode == Keys.Back) klav_Back = true;
        }

        private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (textBox1.Text.Length >= (kol_chel < 5 ? max_dl_m : max_dl_b)
            && klav_Back == false) { e.Handled = true; label5.Visible = true; return;
}
            Display();
        }
    }
```

7. РОБОТА З БАЗАМИ ДАНИХ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ ADO.NET

7.1. Характеристика технології ADO .NET у Visual C#

ADO.NET – новий етап у технології ActiveX Data Objects (ADO, об'єкти даних ActiveX). Якщо раніше в ADO наголос робився на постійне з'єднання з базою даних, то в технології використання ADO.NET спочатку закладено можливість роботи програми в стані "розриву" з'єднання з базою даних. ADO.NET забезпечує можливість роботи з усіма сумісними з OLE DB джерелами даних як у локальних мережах у рамках традиційних Windows-додатків, так і в глобальних мережах (Інтернет) в рамках Web-додатків.

Розглянемо переваги технології ADO.Net над технологією ADO:

1. Від'єднана модель даних. Об'єкт **Connection** потрібно мати лише на етапі з'єднання з базою даних та виконання операцій з нею.

2. Незалежність набору даних від самої бази. При формуванні запиту всі дані містяться у незалежних класах.

Для роботи з ADO.Net передбачено простір імен: **System.Data**, а також **System.Data.OleDb** для роботи з драйверами **OleDB**, **System.Data.Odbc** для роботи з драйверами ODBC, **System.Data.SqlClient** – для роботи з Sql Server.

Назви класів будуть починатися з префікса **OleDB**, **Odbc**, **Sql**.

У технології ADO.Net використовуються наступні об'єкти. Розглянемо призначення кожного з них:

1. Об'єкт **Connection** – призначений лише для з'єднання з БД.

2. Об'єкт **Command** – призначений для реалізації запитів виконання (метод ExecuteNonQuery), тільки запитів на читання з переглядом лише у одну сторону (метод ExecuteReader), підсумкових запитів, які повертають лише одне значення (метод ExecuteScalar).

3. Об'єкт **DataReader** – об'єкт для читання, який можна переглянути лише один раз і в один бік (forward only), призначений для заповнення елементів керування.

4. Об'єкт **DataSet** – колекція наборів записів (об'єктів **DataTable**), що є відображенням всієї БД, з якою працює користувач, або її частини. Кожен об'єкт всередині колекції є набором записів і називається **DataTable** (аналог Recordset у ADO). Даний набір **DataTable** є повноцінним набором та призначений для додавання та зміни вхідних даних.

5. Об'єкт **DataAdapter**. Основне його призначення це виконання запиту до БД на вибірку та заповнення даних набору **DataTable**. Слід зазначити, що одному набору (таблиці) **DataTable** відповідає строго один адаптер **DataAdapter**. Крім формування набору записів основною функцією **DataAdapter** є оновлення даних набору записів в БД. Тобто функція адаптера – це синхронізація та обмін даними між набором та таблицею БД. Для створення набору даних застосовується метод Fill класу **DataAdapter**, оновлення даних у таблиці БД – метод Update.

6. Об’єкт **CommandBuilder**. Для того, щоб оновити дані з незалежного набору в таблицю бази даних, необхідно реалізувати три запити – на оновлення, видалення та додавання. Автоматично ці запити формує об’єкт **CommandBuilder**.

Важливо: Працюючи з таблицями БД, у кожній з них має бути ключове поле. Інакше дані у БД не оновлюватимуться. Наявність ключового поля – обов’язкова умова за нормалізації БД.

На рис. 7.1. показано взаємодію від’єднаної колекції наборів даних **DataSet** з базою даних. Кожна таблиця БД пов’язана з кожною таблицею набору (**DataTable**) за допомогою відповідного адаптера.

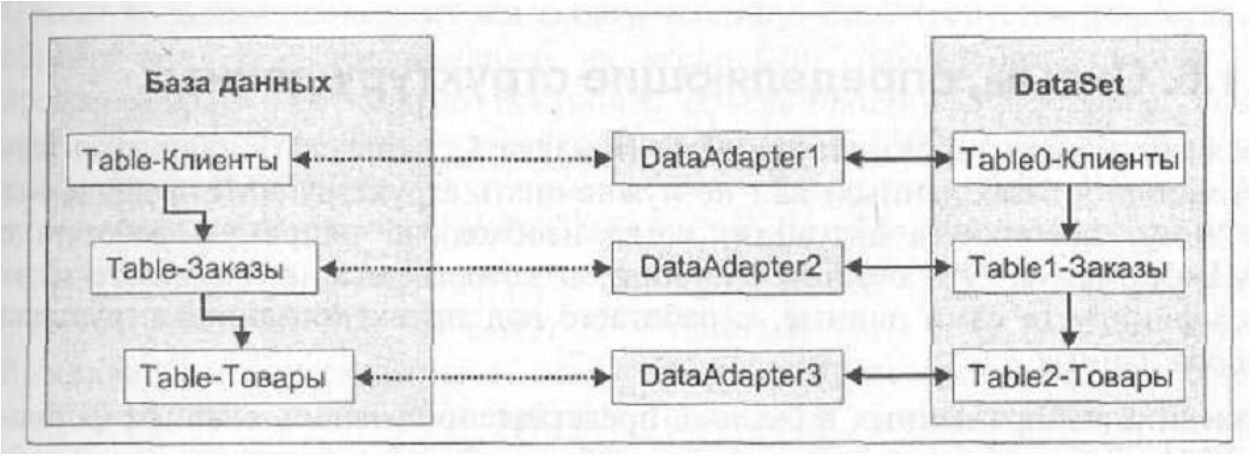


Рис. 7.1. Схема вазаємодії **DataSet** та БД

Для представлення даних у наборі (об’єкт **DataTable**) передбачено об’єкт представлення (**DataView**), в якому дані представлені з урахуванням сортування, фільтрації (тобто те, що ви бачите на екрані). Цей об’єкт можна отримати, використовуючи властивість **DefaultView** об’єкта **DataView**.

Окрім наведених вище об’єктів особливе місце займають об’єкти прив’язки даних і навігації за записами - це об’єкти **BindingSource** і **BindingNavigator**. Об’єкт **BindingSource** призначений для переміщення по конкретному набору, а також для сортування, пошуку та фільтрації записів, оскільки поточний запис, а також методи переміщення в ADO.NET відсутні – це робиться за допомогою елемента керування **BindingSource**.

7.2. Об’єкт Connection

Для з’єднання з базою даних ADO.NET використовується об’єкт **Connection**. Залежно від джерел даних розрізняють такі об’єкти **Connection**:

1. **OleDbConnection** – об’єкт, що дозволяє створити з’єднання з будь-якими джерелами даних через OLE DB. (простір імен **System.Data.Oledb**).

2. **OdbcConnection** – об’єкт, що дозволяє створити з’єднання з будь-якими джерелами даних через ODBC. (простір імен **System.Data.Odbc**).

3. **SqlConnection** – об’єкт, що дозволяє створити з’єднання з базами даних MS SQL Server. (простір імен **System.Data.SqlClient**).

У таблицях 7.1, 7.2 наведено відповідно основні властивості та методи об’єкта **Connection**.

Таблиця 7.1

Властивості об’єкта Connection

Властивості	**Опис**
ConnectionString	Визначає параметри підключення до джерела даних. Цей рядок містить параметри, розділені крапкою з комою. Властивість ConnectionString може задаватися до відкриття з’єднання за допомогою методу Open.
ConnectionTimeout	Встановлює або повертає кількість секунд очікування на підключення до бази даних. Значення за замовчуванням – 15. Доцільно використовувати цю властивість у разі виникнення проблем через щільний мережний трафік або завантаженість сервера. Якщо час, вказаний у ConnectionTimeout, спливає до відкриття підключення, відбувається помилка, і ADO .NET скасовує спробу підключення. Якщо властивості встановити нульове значення, то ADO .NET чекатиме нескінченно, доки підключення не буде відкрито. Необхідно переконатися, що провайдер підтримує властивість ConnectionTimeout. Властивість необхідно встановити до відкриття з’єднання.
Provider	Повертає ім’я постачальника OLE DB, вказаного у рядку підключення.
DataBase	Отримує назву поточної бази даних або бази даних, яка буде використовуватись після відкриття підключення.
DataSource	Повертає ім’я сервера або ім’я джерела даних.
ServerVersion	Повертає рядок, який містить версію сервера, до якого підключається клієнт.
State	Повертає стан об’єкта. Властивість State може приймати такі значення: • ConnectionState.Closed – об’єкт закрито. • ConnectionState.Open – об’єкт відкрито. • ConnectionState.Connecting – об’єкт з’єднується. • ConnectionState.Executing – об’єкт виконує команду. • ConnectionState.Fetching – об’єкт виконує вибірку рядків.

Таблиця 7.2

Методи об’єкту Connection

Методи	**Опис**
Open()	Відкриває підключення до бази даних за допомогою параметрів, заданих у ConnectionString.
Close()	Закриває з’єднання з джерелом даних.
BeginTransaction	Перевантажений. Розпочинає транзакцію бази даних
ChangeDatabase	Змінює поточну базу даних для відкритого Connection
CreateCommand	Створює та повертає об’єкт Command, пов’язаний з Connection
GetSchema(String)	Повертає відомості схеми для джерела даних цього об’єкта Connection, використовуючи вказаний рядок для імені схеми.
GetSchema(String, String[])	Повертає відомості схеми для джерела даних цього об’єкта Connection, використовуючи вказаний рядок для імені схеми та вказаний масив рядків для значень обмежень.

Клас **Connection** керує інформацією, необхідною для підключення до постачальника джерела даних і дозволяє відкривати та закривати активне з'єднання з ним за допомогою методів ***Open*** та ***Close***. Крім того, можна створювати транзакції за допомогою методу **BeginTransaction**. Виконання та подальша робота з транзакціями реалізується за допомогою класу **Command**.

7.3. Об'єкт Command

Об'єкт **Command** використовується для реалізації запитів на виконання, а також запитів на вибірку, які виконуються на стороні сервера та повертають результат лише для читання.

Об'єкт **Command** створюється з урахуванням існуючого з'єднання.

У таблицях 7.3 та 7.4 наведено відповідно основні властивості та методи об'єкта **Command**.

Таблиця 7.3

Властивості об'єкта Command

Властивості	**Опис**
Connection	Визначає об'єкт Connection, з яким пов'язаний об'єкт Command.
CommandText	Рядок, що визначає текст команди, наприклад, оператор SQL, ім'я таблиці або виклик процедури, що зберігається.
CommandTimeout	Встановлює або повертає кількість секунд очікування на виконання команди. Значення за замовчуванням – 30. Необхідно використовувати цю властивість у разі виникнення проблем через щільний мережний трафік або завантаженість сервера. Якщо час, вказаний у CommandTimeout, спливає до завершення виконання команди, відбувається помилка, і ADO.NET скасовує команду. Якщо властивості встановлено нульове значення, то ADO.NET чекатиме нескінченно доти, доки команда не буде виконана.
CommandType	Визначає тип команди. Можливі значення: • CommandType.StoredProcedure – процедура, що зберігається. • CommandType.Text – текстове визначення команди або процедури, що зберігається. • CommandType.TableDirect – створення SQL-запиту, який повертає всі рядки вказаної таблиці.
Parameters	Повертає набір ParameterCollection (для роботи з параметричними запитами).
Transaction	Повертає або задає транзакцію Transaction, у якій виконується команда Command.

Таблиця 7.4

Методи об’єкта Command

Методи	Опис
ExecuteNonQuery()	Запускає SQL-запит на виконання без повернення результуючого набору (INSERT INTO, DELETE, UPDATE тощо) і повертає кількість рядків на які вплинув запит.
ExecuteReader()	Виконує SQL-запит на вибірку та повертає об’єкт DataReader тільки для читання та заповнення елементів керування. Об’єкт DataReader можна прочитати лише один раз зверху донизу.
ExecuteScalar()	Виконує запит та повертає перший стовпець першого рядка результуючого набору, що повертається запитом. Додаткові стовпці та рядки ігноруються. Призначений для реалізації підсумкових запитів, результатом яких буде одне число (наприклад, вибірка з використанням функцій: SUM(), COUNT(), AVG() і т.д.).
Cancel()	Скасовує виконання запиту, який реалізується за допомогою об’єкта Command.
CreateParameter	Створює та повертає об’єкт класу Parameter.

7.4. Об’єкт DataAdapter

Основне призначення об’єкта **DataAdapter** – це виконання запиту до БД на вибірку та заповнення даних набору **DataTable**. Слід зазначити, що одному набору (таблиці) **DataTable** відповідає виключно один адаптер **DataAdapter**. Крім формування набору записів основною функцією **DataAdapter** є оновлення даних набору записів в БД. Тобто функція адаптера – це синхронізація та обмін даними між набором та таблицею БД. Для створення набору даних застосовується метод **Fill** класу **DataAdapter**, а для оновлення даних у таблиці БД – метод **Update**.

У таблицях 7.5, 7.6 наведено відповідно основні властивості та методи об’єкта **DataAdapter**.

Таблиця 7.5

Властивості об’єкту DataAdapter

Властивості	Опис
DeleteCommand	Повертає або задає оператор SQL або процедуру, що зберігається, для видалення записів з набору даних.
InsertCommand	Повертає або задає оператор SQL або процедуру, що зберігається, для додавання нових записів у джерело даних.
UpdateCommand	Повертає або задає оператор SQL або процедуру, що зберігається, для оновлення записів у джерелі даних.
SelectCommand	Повертає або задає оператор SQL або процедуру, що зберігається, для вибірки записів у джерелі даних.
TableMappings	Отримує колекцію, що забезпечує основне зіставлення між вихідною таблицею та DataTable.

Таблиця 7.6

Методи об’єкта DataAdapter

Методи	Опис
Fill(DataSet)	Заповнює дані об’єкта DataTable всередині колекції DataSet під час реалізації SQL-запиту на вибірку. У цьому методі як вхідний параметр може бути не лише колекція таблиць DataSet, а й сама таблиця – об’єкт DataTable.
Update(DataSet)	Викликає відповідні оператори INSERT, UPDATE або DELETE для кожного створеного, оновленого або видаленого рядка в об’єкті DataSet. У цьому методі як вхідний параметр може бути не лише колекція таблиць DataSet, а й сама таблиця – об’єкт DataTable.

7.5. Об’єкт DataSet

Об’єкт **DataSet** є колекцією наборів даних – об’єктів **DataTable**. За своєю суттю об’єкт **DataSet** – це модель бази даних, що включає певні таблиці.

У таблицях 7.7 та 7.8 наведено відповідно основні властивості та методи об’єкта **DataSet**.

Таблиця 7.7

Вдастивості об’єкта DataSet

Властивості	Опис
Tables	Повертає колекцію таблиць класу DataSet.
Relations	Повертає колекцію співвідношень, що зв’язують таблиці та дозволяють переходити від батьківських таблиць до дочірніх.

Таблиця 7.8

Методи об’єкта DataSet

Методи	Опис
AcceptChanges()	Зберігає всі зміни, внесені до класу DataSet після завантаження або після останнього виклику методу AcceptChanges.
RejectChanges()	Скасовує всі зміни, внесені до класу DataSet після його створення або після останнього виклику методу DataSet.AcceptChanges.
Clear()	Видаляє з DataSet будь-які дані шляхом видалення всіх рядків у всіх таблицях.
GetXml()	Повертає представлення даних XML, що зберігаються в класі DataSet.
ReadXml(String)	Зчитує XML-схему та дані у DataSet, використовуючи вказаний файл.
WriteXml(String, XmlWriteMode)	Записує поточні дані та по можливості схему для DataSet у вказаний файл за допомогою заданого об’єкта XmlWriteMode. Щоб записати схему, вкажіть у параметрі mode значення WriteSchema.

7.6. Об'єкт DataTable

Об'єкт **DataTable** є набором записів, як результатів вибірки SQL-запиту.

У таблицях 7.9 та 7.10 наведено відповідно основні властивості та методи об'єкта **DataTable**.

Таблиця 7.9

Властивості об'єкта DataTable

Властивості	**Опис**
ChildRelations	Отримує колекцію дочірніх відносин для DataTable.
Rows	Отримує колекцію рядків, що належать даній таблиці.
Columns	Отримує колекцію стовпців, що належать даній таблиці.
Constraints	Отримує колекцію обмежень, що містяться у цій таблиці.
DataSet	Отримує клас DataSet, до якого належить дана таблиця.
DefaultView	Отримує налаштування таблиці (об'єкт DataView), яке може включати в себе подання з фільтром або положення курсору.
PrimaryKey	Повертає або задає масив стовпців, які є стовпцями первинного ключа для таблиці даних.
TableName	Повертає або вказує ім'я DataTable.

Таблиця 7.10

Методи об'єкта DataTable

Методи	**Опис**
AcceptChanges()	Фіксує всі зміни, внесені до таблиці після останнього виклику методу AcceptChanges.
RejectChanges()	Виконується відкат усіх змін, внесених до таблиці з моменту завантаження або після останнього виклику методу AcceptChanges.
Clear()	Очищає DataTable від усіх даних.
ReadXml(String)	Читає дані та схему XML у DataTable із зазначеного файлу.
WriteXml(String, XmlWriteMode)	Записує поточні дані та по можливості схему для DataTable, використовуючи вказаний файл та заданий перелік XmlWriteMode. Щоб записати схему, вкажіть у параметрі mode значення WriteSchema.
NewRow()	Створює новий об'єкт класу рядка DataRow, що має ту саму структуру, що й таблиця.
Select()	Отримує масив всіх об'єктів DataRow.
Select(String)	Отримує масив всіх об'єктів DataRow, які відповідають умовам фільтру.
Select(String, String)	Отримує масив всіх об'єктів DataRow, що відповідають умовам фільтра, згідно з вказаним порядком сортування.
Select(String, String, DataViewRowState)	Отримує масив всіх об'єктів DataRow, що відповідають умовам фільтра, згідно з порядком сортування, що відповідає зазначеному стану.

7.7. Об'єкт DataReader

Об'єкт **DataReader** є набором записів доступних тільки для читання, отриманим в результаті виконання методу **ExecuteReader** об'єкта **Command**. Цей об'єкт призначений для заповнення масивів даних та елементів керування лише один раз.

У таблицях 7.11 та 7.12 наведено відповідно основні властивості та методи об'єкта **DataReader**.

Таблиця 7.11

Властивості об'єкта DataReader

Властивості	**Опис**
FieldCount	Повертає кількість стовпців у поточному рядку.
HasRows	Отримує значення, яке вказує на те, чи об'єкт DataReader містить один або кілька рядків.
IsClosed	Вказує, чи пристрій зчитування даних закрито.
Item[Int32]	Повертає значення вказаного стовпця у власному форматі за наявності заданого порядкового номера стовпця.
Item[String]	Повертає значення зазначеного стовпця із заданим ім'ям у власному форматі.

Таблиця 7.12

Методи об'єкта DataReader

Методи	**Опис**
Read()	Переміщує об'єкт класу DataReader до наступного запису.
GetSchemaTable()	Повертає об'єкт DataTable, який описує метадані стовпців об'єкта DataReader.
GetName()	Повертає ім'я заданого стовпця.
GetOrdinal()	Повертає порядковий номер стовпця за наявності заданого імені стовпця.

7.8. Об'єкт CommandBuilder

Об'єкт **CommandBuilder** автоматично генерує однотабличні команди (INSERT INTO, DELETE, UPDATE), які дозволяють вносити зміни з об'єкта **DataTable** до таблиці бази даних. Ці зміни виконуються за допомогою методу **Update** об'єкта **DataAdapter**.

У таблицях 7.13 та 7.14 наведено відповідно основні властивості та методи об'єкта **CommandBuilder**.

Таблиця 7.13

Властивості об’єкта CommandBuilder

Властивості	Опис
DataAdapter	Повертає або задає об’єкт DataAdapter, для якого автоматично створюються оператори SQL.
QuotePrefix	Повертає або задає початковий символ або символи, які використовуються для вказування об’єктів бази даних (наприклад, таблиць або стовпців), імена яких містять символи, такі як пробіли або зарезервовані слова. Рекомендується як початковий символ встановити «[».
QuoteSuffix	Отримує або задає кінцевий символ або символи, які використовуються для вказування об’єктів бази даних (наприклад, таблиць або стовпців), імена яких містять символи, такі як пробіли або зарезервовані слова. Рекомендується як кінцевий символ встановити «]».

Таблиця 7.14

Методи об’єкта CommandBuilder

Методи	Опис
GetDeleteCommand()	Отримує об’єкт Command, що автоматично створюється, та який потрібний для виконання операцій видалення в джерелі даних.
GetInsertCommand()	Отримує об’єкт Command, що автоматично створюється, та який потрібний для виконання операцій вставки в джерелі даних.
GetUpdateCommand()	Повертає об’єкт Command, що автоматично створюється, та який потрібний для виконання операцій оновлення в джерелі даних.

7.9. Об’єкт DataView

Об’єкт **DataView** є представленням даних об’єкта таблиці **DataTable**. Сортування та фільтрація даних не впливають на таблицю, а лише на подання її даних, яке може бути прив’язане до того чи іншого елемента керування. Отримати представлення даних можна за допомогою властивості **DefaultView** об’єкта **DataTable**.

У таблицях 7.15, 7.16 наведено відповідно основні властивості та методи об’єкта **DataView**.

Таблиця 7.15

Властивості об’єкта DataView

Властивості	Опис
AllowDelete	Задає або отримує значення, яке показує, чи дозволено виконувати видалення.
AllowEdit	Повертає або задає значення, що показує, чи можна виконувати оновлення.
AllowNew	Повертає або задає значення, яке визначає можливість додавання нових рядків за допомогою методу AddNew.
ApplyDefaultSort	Повертає або задає значення, яке визначає необхідність використання сортування за замовчуванням.
Count	Отримує кількість записів у DataView після застосування властивостей RowFilter та RowStateFilter.

Продовження таблиці 7.15

Властивості	Опис
Item	Отримує рядок даних із зазначеної таблиці.
RowFilter	Повертає або задає вираз, який використовується для вибору рядків, які переглядаються в об’єкті DataView.
RowStateFilter	Повертає або задає фільтр стану рядків, який використовується в DataView.
Sort	Повертає або задає стовпець або стовпці для сортування, а потім – порядок сортування для DataView.
Table	Повертає або задає вихідний об’єкт DataTable.

Таблиця 7.16

Методи об’єкта DataView

Методи	Опис
AddNew()	Додає новий рядок у DataView.
Delete()	Видаляє рядок за вказаним індексом.
Find(Object)	Знаходить рядок у DataView за вказаним значенням ключа сортування.
Find(Object[])	Знаходить рядок DataView за вказаними значеннями ключа сортування.

7.10. Об’єкт BindingSource

Об’єкт **BindingSource** є універсальним зв’язувачем набору записів (об’єкта **DataTable**) та довільних елементів керування, таких як **DataGridView**, **BindingNavigator** (рядок навігації за записами) тощо. Крім функцій універсального зв’язувача, об’єкт **BindingSource** має функції переходу за записами, а також функції фільтрації та сортування даних. Крім того, в даному об’єкті запам’ятовується поточна позиція набору даних.

У таблицях 7.17 та 7.18 наведено відповідно основні властивості та методи об’єкта **BindingSource**.

Таблиця 7.17

Властивості об’єкта BindingSource

Властивості	Опис
AllowEdit	Повертає значення, яке показує, чи можна редагувати елементи в наборі записів.
AllowNew	Повертає або задає значення, яке вказує, чи може бути використаний метод AddNew для додавання елементів до набору.
AllowRemove	Повертає значення, яке показує, чи можна видаляти елементи з набору.
Count	Повертає загальну кількість елементів у базовому списку, відфільтрованих відповідно до значення властивості Filter.
Current	Отримує поточний елемент у наборі записів.
DataMember	Повертає або задає список у джерелі даних, до якого зараз прив’язаний з’єднувач.
DataSource	Повертає або задає джерело даних, до якого прив’язаний з’єднувач.
Filter	Повертає або задає вираз, що використовується для фільтрації рядків, що переглядаються.

Продовження таблиці 7.17

Властивості	Опис
Sort	Повертає або вказує імена стовпців, які використовуються для сортування, та порядок сортування для перегляду рядків у джерелі даних.
SortDirection	Отримує напрямок сортування елементів у наборі.
Position	Повертає або задає індекс поточного елемента набору даних.

Таблиця 7.18

Методи об’єкта BindingSource

Методи	Опис
AddNew()	Додає новий елемент до набору даних.
Clear()	Видаляє всі елементи набору.
Find(String, Object)	Повертає індекс елемента у наборі, що має задане ім’я властивості та значення.
Insert()	Вставляє елемент у список за вказаним індексом.
MoveFirst()	Перехід на перший запис.
MoveLast()	Перехід на останній запис.
MoveNext()	Перехід до наступного запису.
MovePrevious()	Перехід на попередній запис.
Remove()	Видалення заданого елемента.
RemoveAt()	Видалення елемента за вказаним індексом.
RemoveCurrent()	Видалення поточного елемента.
RemoveFilter()	Видаляє фільтр, зіставлений з компонентом BindingSource.
RemoveSort()	Видаляє сортування, зіставлене з компонентом BindingSource.

Приклади

7.11. Приклад ADO1

У прикладі **ADO1** використовується база даних "nwind.mdb". У цьому прикладі розглядається таблиця ***Categories***, в якій описані категорії товарів, їх найменування. Ця таблиця є довідковою. Вона пов’язується відношенням «один до багатьох» з таблицею ***Products***, у якій вказується характеристика конкретного виду продукції з певної категорії (таблиця ***Categories***). Ключове поле для зв’язування – ***CategoryID***. Схему даних наведено на рис. 7.2.

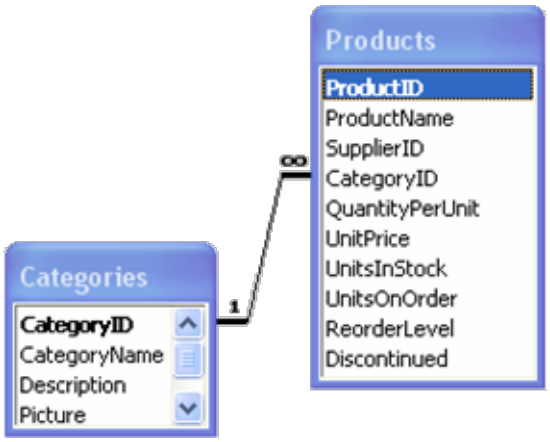


Рис. 7.2. Схема даних БД "nwind.mdb" для реалізації прикладу ADO1

Мета цього завдання – отримати список товарів конкретної категорії. Ця програма зроблена на основі об’єкта для читання **DataReader**. Далі наведено текст цієї програми:

```
using System.Data;
using System.Data.OleDb;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Ado1
{
    public partial class Form1 : Form
    {
        //Объект Connection
        OleDbConnection conn;

        public Form1()
        {
            InitializeComponent();

            //Создание объекта Connection и Recordset
            try
            {
      string source = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=NWIND.MDB";
           string select = "SELECT CategoryName, CategoryID FROM Categories";
                conn = new OleDbConnection(source);
                conn.Open();

                OleDbCommand cmd = new OleDbCommand(select, conn);
                OleDbDataReader reader = cmd.ExecuteReader();

                //Заполнение 1-го листбокса
                while(reader.Read())
                    listBox1.Items.Add(new Name(reader[0].ToString(),
                                        (int)reader[1]));
                reader.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            string str = "SELECT ProductName FROM Products WHERE CategoryID="+
                         ((Name)listBox1.SelectedItem).ItemData.ToString();

            OleDbCommand cmd = new OleDbCommand(str, conn);
            OleDbDataReader reader = cmd.ExecuteReader();

            listBox2.Items.Clear();
            //Заполнение 2-го листбокса
            while (reader.Read())
                listBox2.Items.Add(reader[0]);
```

```
            reader.Close();
        }
    }

    //Класс для заполнения категорий продукции с итемдатой (для 1-го листбокса)
    class Name
    {
        public Name(string n, int I)
        {
            name = n;
            ItemData = I;
        }

        public override string ToString() { return name; }

        public string name;
        public int ItemData;

    }

}
```

У цьому прикладі у конструкторі класу **Form1()** спочатку відбувається підключення до БД, а потім заповнюється елемент керування **ListBox** категоріями продукції. Нижче наведено фрагмент програми:

```
//Создание объекта Connection и Recordset
            try
            {
     string source = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=NWIND.MDB";
           string select = "SELECT CategoryName, CategoryID FROM Categories";

                conn = new OleDbConnection(source);
                conn.Open();

                OleDbCommand cmd = new OleDbCommand(select, conn);
                OleDbDataReader reader = cmd.ExecuteReader();

                //Заполнение 1-го листбокса
                while(reader.Read())
                    listBox1.Items.Add(new Name(reader[0].ToString(),
                                    (int)reader[1]));
                reader.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

Для підключення БД використовується **OLE DB**, тому для підключення та роботи всі класи починаються з префікса OleDb. Як рядок підключення до класу **OleDbConnection** передамо наступний рядок підключення:

```
string source = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=NWIND.MDB"
```

Після цього відкриємо з’єднання за допомогою методу **Open**.

Далі створимо об’єкт **OledbCommand**, який прив’язаний до нашого об’єкта-з’єднання та передамо йому текст SQL-запиту з іменами та ідентифікаторами категорій товарів.

Виконується цей запит за допомогою методу **ExecuteReader()** і повертається об’єкт для читання записів **OleDbDataReader**.

На наступному етапі за допомогою методу **Read** читаємо кожен запис набору і записуємо все в елемент керування **listBox1** класу **ListBox**.

Після формування списку, об’єкт **OleDbDataReader** закривається методом **Close()**.

Слід зазначити, що елемент керування **ListBox** у C# є контейнером довільних об’єктів. З цією метою було створено клас **Name** для зберігання категорій продукції, і навіть їх ідентифікаторів.

```
//Класс для заполнения категорий продукции с итемдатой (для 1-го листбокса)
class Name
{
    public Name(string n, int I)
    {
        name = n;
        ItemData = I;
    }
    public override string ToString() { return name; }
    public string name;
    public int ItemData;
}
}
```

Перевантажений метод **ToString()** у класі говорить про те, які саме дані відображатимуться у списку. У нашому випадку відображатиметься змінна **name** типу **string** (категорія продукції).

У наступній процедурі **listBox1_SelectedIndexChanged()** відображаються дії, які відбуваються, коли користувач натискає на одну з категорій товарів. У цьому випадку по індексному полю **CategoryID**, яке у **listBox1**, буде виведено найменування всіх товарів з таблиці **Products** бази даних **nwind.mdb**. Усі вони записуються в об’єкт **listBox2** класу **ListBox**.

Далі наведено функцію відгуку **listBox1_SelectedIndexChanged()**:

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    string str = "SELECT ProductName FROM Products WHERE CategoryID="+
                  ((Name)listBox1.SelectedItem).ItemData.ToString();
    OleDbCommand cmd = new OleDbCommand(str, conn);
    OleDbDataReader reader = cmd.ExecuteReader();
    listBox2.Items.Clear();
    //Заполнение 2-го листбокса
    while (reader.Read())
        listBox2.Items.Add(reader[0]);
    reader.Close();
}
}
```

У даному фрагменті в змінній **str** типу **string** формується запит за значенням поля **CategoryID**, яке повертається з **listbox** (поле **ItemData**).

Таким чином, формується SQL-запит на вибірку із заданою умовою щодо конкретної категорії товарів. Результатом запиту є об'єкт-читач **OleDbDataReader** з даними про найменування товарів (поле **ProductName** таблиці **Products**). За допомогою методу **Clear()** листбокс постійно очищається та заповнюється записами об'єкта **OleDbDataReader**, аналогічно першому фрагменту програми (заповнення записами **listbox m_category_prod**). Наприкінці фрагмента об'єкт **OleDbDataReader** закривається (метод **Close()**).

Результат виконання програми **ADO1** наведено на рис. 7.3.

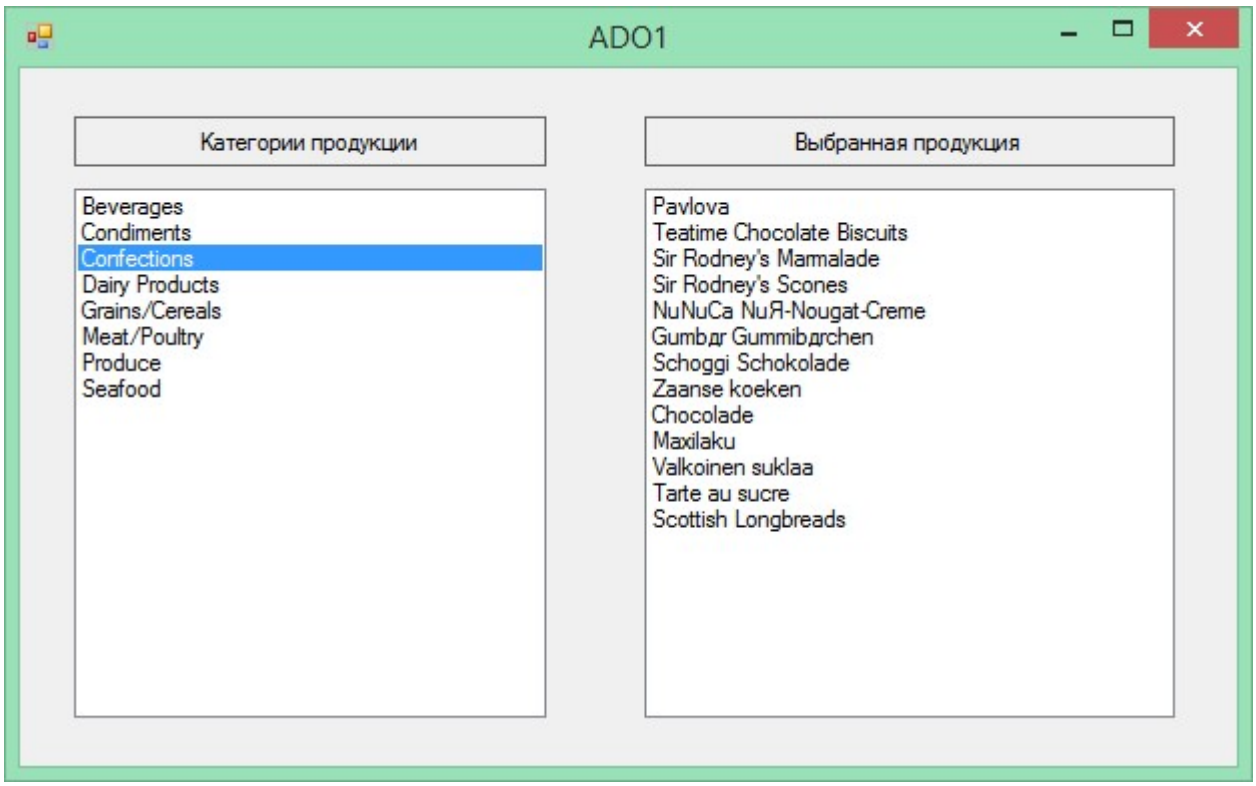


Рис. 7.3. Результат роботи програми ADO1

7.12. Приклад ADO2

У цьому прикладі продемонструємо початкову роботу з довільною базою даних та прив'язкою її набору даних до візуальних елементів (**DatagridView**, **ListBox**, **TextBox**). У цьому випадку набір даних з бази даних буде створюватися з використанням об'єкта **DataAdapter**.

Крім того, у даному прикладі розглянемо можливість контролю даних таблиці, вбудованого в елементі керування **DatagridView** та реалізованого за допомогою класу **ErrorProvider**.

Для початку роботи в конструкторі форм **Designer** спроектуємо форму, де розмістимо відповідні елементи керування (рис. 7.4):

1. **button1** – кнопка вибору БД.
2. **button2** – кнопка запиту до БД.
3. **button3** – кнопка оновлення даних БД.
4. **textBox1** – текстове поле для введення запиту.
5. **openFileDialog1** – діалог вибору БД.
6. **bindingSource1** – універсальний об’єкт зв’язувач з набором даних.
7. **bindingNavigator1** – об’єкт навігації (переміщення по записам).
8. **dataGridView1** – об’єкт візуального відображення даних набору.
9. **listBox1** – список для відображення даних поля.
10. **textBox1** – текстове поле відображення вибраного елемента поля.

За допомогою властивостей **Dock** та **Anchor** елементів керування здійснюємо прив’язку візуальних елементів.

Вся робота з базами даних відбуватиметься у програмному режимі, що дозволяє універсально працювати з довільною БД та зв’язувати дані з візуальними елементами.

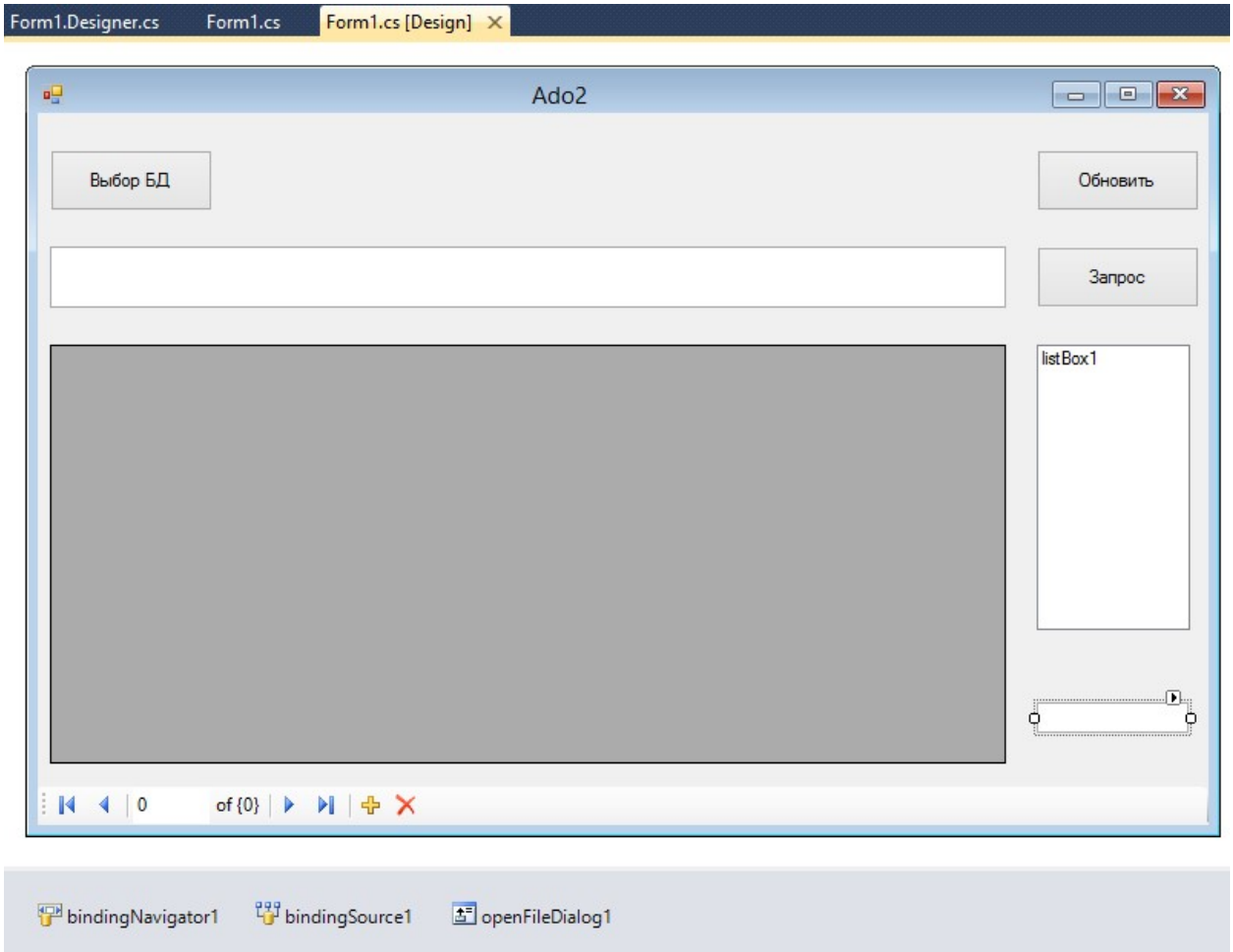


Рис. 7.4. Конструювання додатку ADO2

Далі наведено повний текст програмного коду прикладу ADO2:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OleDb;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Ado22
{
    public partial class Form1 : Form
    {
        //Объект Connection с драйвером OleDb
        OleDbConnection conn;
        OleDbDataAdapter da;
        DataSet ds;

        public Form1()
        {
            InitializeComponent();
            conn = null;
            da = null;
            ds = null;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            //Открытие файла *.mdb
            openFileDialog1.FileName = "otl.mdb";
            openFileDialog1.Filter = "*.mdb|*.mdb";
            if (openFileDialog1.ShowDialog() != DialogResult.OK) return;

            //Формирование объекта Connection
            try
            {
                string source = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
                                openFileDialog1.FileName;
                if (conn != null) conn.Dispose();
                conn = new OleDbConnection(source);
                conn.Open();
                textBox1.Text = "Select * From otl_tab";
            }

            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            try
            {
                //Создание объекта Recordset
                if (da != null) da.Dispose();
                da = new OleDbDataAdapter(textBox1.Text, conn);

                //Сохранение данных через объект OleDbCommandBuilder
                OleDbCommandBuilder bulder = new OleDbCommandBuilder(da);
                //Символы в которые будут заключены поля и таблицы БД
                //при формировании запросов
```

```
        bulder.QuotePrefix = "[";
        bulder.QuoteSuffix = "]";

        if (ds != null) ds.Dispose();
        ds = new DataSet();
        da.Fill(ds);

        //Отсутствие пустых значений
        ds.Tables[0].Columns[2].AllowDBNull = false;
        //Значение по умолчанию
        ds.Tables[0].Columns[0].DefaultValue = 20;

        //Привязка данных
        bindingSource1.DataSource = ds.Tables[0];
        dataGridView1.DataSource = bindingSource1;
        bindingNavigator1.BindingSource = bindingSource1;
        //Привязка столбца к листбоксу и текстбоксу
        listBox1.DataSource = bindingSource1;
        listBox1.DisplayMember = "id";
        textBox2.DataBindings.Add("Text", bindingSource1, "id");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private void button3_Click(object sender, EventArgs e)
{
    try
    {
        da.Update(ds.Tables[0]);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }

}

//Отклик для поиска стандартных ошибок при работе с данными
private void dataGridView1_DataError(object sender,
              DataGridViewDataErrorEventArgs e)
{
    dataGridView1.Rows[e.RowIndex].ErrorText = e.Exception.Message;
    e.Cancel = true;
}

//Проверка ячейки (отклик на потерю фокуса)
private void dataGridView1_CellValidating(object sender,
DataGridViewCellValidatingEventArgs e)
{
    try
    {
         /*
         e.ColumnIndex - индекс поля
         e.RowIndex - индекс строки
         e.FormattedValue - значение
         */
        if (dataGridView1.Columns[e.ColumnIndex].Name.ToUpper() == "KOD")
        {
            if (Convert.ToInt32(e.FormattedValue) < 1 ||
                        Convert.ToInt32(e.FormattedValue) > 200)
            {
```

```
                        dataGridView1.Rows[e.RowIndex].ErrorText =
                            "Значение поля код должно лежать в диапазоне от 1 до 200";
                        e.Cancel = true;
                    }
                }
            }
            catch (Exception ex)
            {
                dataGridView1.Rows[e.RowIndex].ErrorText = ex.Message;
                e.Cancel = true;
            }
        }
        //Завершение редактирования ячейки
       private void dataGridView1_CellEndEdit(object sender, DataGridViewCellEventArgs e)
        {
            dataGridView1.Rows[e.RowIndex].ErrorText = "";
        }

    }
}
```

Для роботи з **OleDb** необхідно підключити простір імен **System.Data.OleDb**.

Для реалізації цього прикладу в класі форми оголошуємо три об'єкти: **conn** класу **OleDbConnection**, **da** класу **OleDbDataAdapter**, **ds** класу **DataSet**.

Об'єкт **OleDbConnection** призначений для з'єднання з базою даних, **OleDbDataAdapter** – для здійснення взаємодії між таблицею бази даних на стороні сервера та набором даних на стороні клієнта (об'єкта **DataTable**, який є елементом контейнера **DataSet**). Так, насамперед за допомогою методу **Fill()** об'єкта **OleDbDataAdapter** ми отримуємо набір даних на стороні клієнта, а потім за допомогою методу **Update()** – надсилаємо запити на видалення, оновлення та додавання даних на сервер.

При натисканні на кнопку «Вибір БД» спрацьовує відгук **button1_Click**, у якому вибирається база даних ***.mdb** та формується її рядок підключення. З використанням об'єкта **Connection** відкриваємо довільну базу даних (метод **Open()**).

При натисканні на кнопку «Запит» спрацьовує відгук **button2_Click**, у якому за допомогою об'єкта **DataAdapter** заповнюється об'єкт **DataTable** контейнера **DataSet** за заданим у полі **textBox1** SQL-запитом.

Нижче наведено рядки формування набору даних:

```
if (da != null) da.Dispose();
da = new OleDbDataAdapter(textBox1.Text, conn);
// Сохранение данных через объект OleDbCommandBuilder
OleDbCommandBuilder bulder = new OleDbCommandBuilder(da);
//Символы в которые будут заключены поля и таблицы БД
//при формировании запросов
bulder.QuotePrefix = "[";
bulder.QuoteSuffix = "]";
if (ds != null) ds.Dispose();
ds = new DataSet();
da.Fill(ds);
```

Вхідними даними в конструкторі об'єкта **OleDbDataAdapter** виступають SQL-запит, написаний у текстовому полі, і поточне відкрите з'єднання conn.

Важливим у даному фрагменті є створення класу **OleDbCommandBuilder**, який прив'язується до об'єкта **OleDbDataAdapter** і призначений для формування трьох запитів при надсиланні змінених даних у таблиці на стороні клієнта на сервер до бази даних. Це запити на видалення, оновлення та додавання даних до таблиці (DELETE, UPDATE, INSERT INTO). При цьому важливо вказати символи квадратних дужок «[» та «]» відповідно до властивостей **QuotePrefix** і **QuoteSuffix** об'єкта **OleDbCommandBuilder**. Так, якщо поля таблиці БД будуть містити прогалини або зарезервовані слова, їх необхідно взяти у квадратні дужки, інакше виникне помилка виконання запитів.

Заповнення першої таблиці об'єкті **DataSet** з БД здійснюється з допомогою методу **Fill()**.

Наступні рядки здійснюють прив'язку даних через універсальний об'єкт-зв'язувач **BindingSource**.

```
.....................................................................
            //Привязка данных
            bindingSource1.DataSource = ds.Tables[0];
            dataGridView1.DataSource = bindingSource1;
            bindingNavigator1.BindingSource = bindingSource1;

            //Привязка столбца к листбоксу и текстбоксу
            listBox1.DataSource = bindingSource1;
            listBox1.DisplayMember = "id";
            textBox2.DataBindings.Add("Text", bindingSource1, "id");
.....................................................................
```

Так, один раз прив'язавши обрану таблицю до об'єкта **BindingSource**, можна прив'язувати цей елемент до будь-яких елементів керування. Прив'яжемо об'єкт **bindingSource1** до об'єкта **dataGridView1** і **bindingNavigator1**.

Також у фрагменті показано варіанти прив'язки об'єкта **bindingSource1** до елемента **ListBox** та елемента **TextBox**.

При натисканні на кнопку «Оновити» спрацьовує відгук **button3_Click**, де оновлюються дані в БД на стороні сервера. Тобто всі зміни у таблиці на стороні клієнта відобразяться у базі даних. Це робиться за рахунок трьох запитів, створених об'єктом **CommandBuilder**.

Рядок відгуку на оновлення даних виглядає так:

```
da.Update(ds.Tables[0]);
```

На рис. 7.5. наведено результат виконання програми ADO2.

При реалізації будь-якого завдання важливим етапом є контроль вихідних даних.

В елемент керування **DataGridView** вбудовано клас **ErrorProvider**, за допомогою якого можна контролювати вихідні дані.

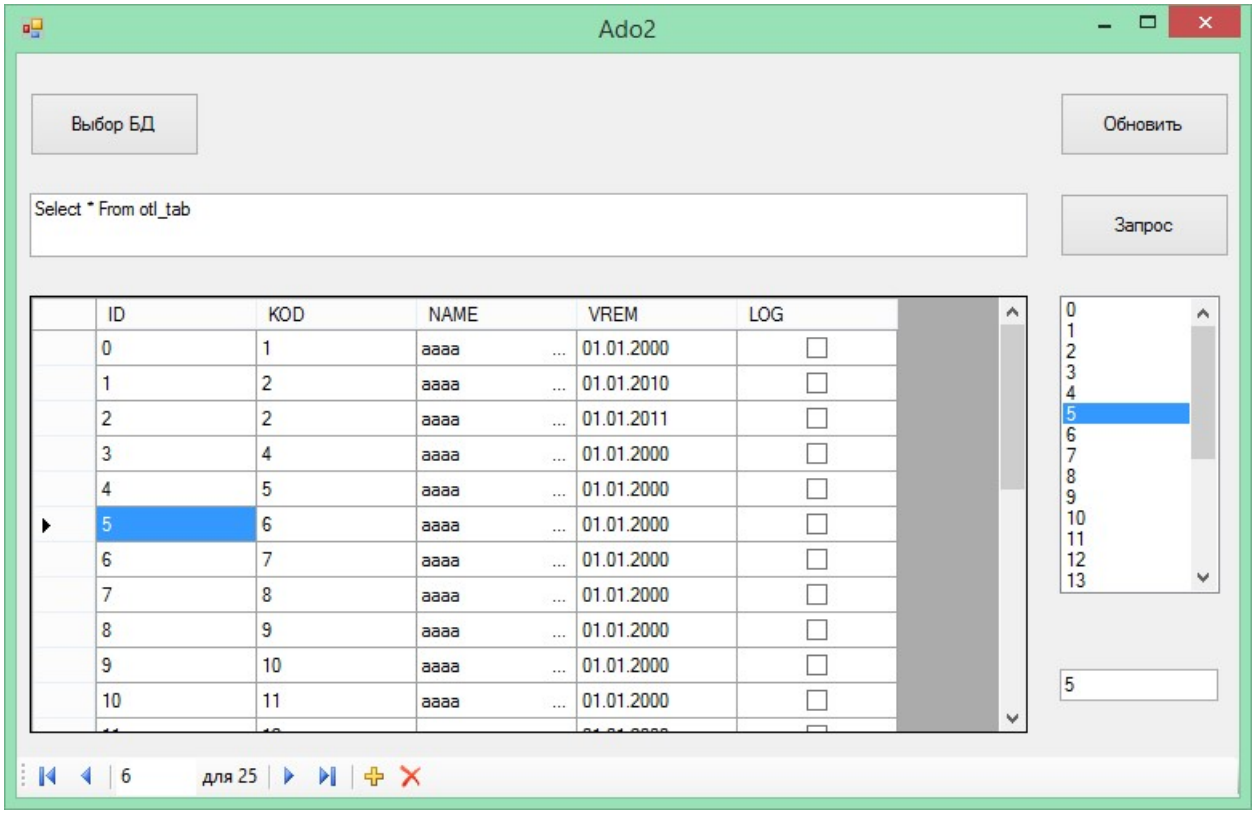


Рис. 7.5. Результат роботи програми ADO2

Для цього зробимо відгук на подію **CellValidating**, яка спрацьовує, коли комірка таблиці втрачає свій фокус:

```
//Проверка ячейки (отклик на потерю фокуса)
private void dataGridView1_CellValidating(object sender,
DataGridViewCellValidatingEventArgs e)
{
    try
    {
         /*
         e.ColumnIndex - индекс поля
         e.RowIndex - индекс строки
         e.FormattedValue - значение
         */

        if (dataGridView1.Columns[e.ColumnIndex].Name.ToUpper() == "KOD")
        {
            if (Convert.ToInt32(e.FormattedValue) < 1 ||
                        Convert.ToInt32(e.FormattedValue) > 200)
            {
                dataGridView1.Rows[e.RowIndex].ErrorText =
                    "Значение поля код должно лежать в диапазоне от 1 до 200";
                e.Cancel = true;
            }
        }
    }
    catch (Exception ex)
    {
        dataGridView1.Rows[e.RowIndex].ErrorText = ex.Message;
        e.Cancel = true;
    }
}
```

У цю подію приходить об’єкт **e** класу **DataGridViewCellValidatingEventArgs**, в якому знаходяться властивості індексу поля, індексу рядка, а також значення комірки, що змінюється.

У функції перевіряється діапазон значень конкретного поля та у разі помилки, властивості **ErrorText** об’єкта **Row** привласнюється повідомлення. Також властивості **Cancel** об’єкта **e** присвоюється значення **true**, що не дає втратити фокус даної комірки доти, доки не буде введено правильне значення.

Крім помилок користувача можна обробляти стандартні помилки. Усі стандартні помилки під час роботи з БД, такі як невідповідність формату даних, переповнення типів тощо, можна зловити у випадку **DataError**.

Наведемо текст цієї функції:

```
//Отклик для поиска стандартных ошибок при работе с данными
private void dataGridView1_DataError(object sender,
              DataGridViewDataErrorEventArgs e)
{
    dataGridView1.Rows[e.RowIndex].ErrorText = e.Exception.Message;
    e.Cancel = true;
}
```

У даному методі можна одразу привласнити властивості **ErrorText** об’єкта **Row** повідомлення про помилку.

При виникненні помилок навпроти помилкового рядка буде розташовано елемент зі знаком вигуку «!» при наведенні на який можна прочитати текст повідомлення (рис. 7.6). При виникненні помилки можна або виправити помилку та натиснути клавішу **Enter** або скасувати зміни, натиснувши клавішу **Esc**.

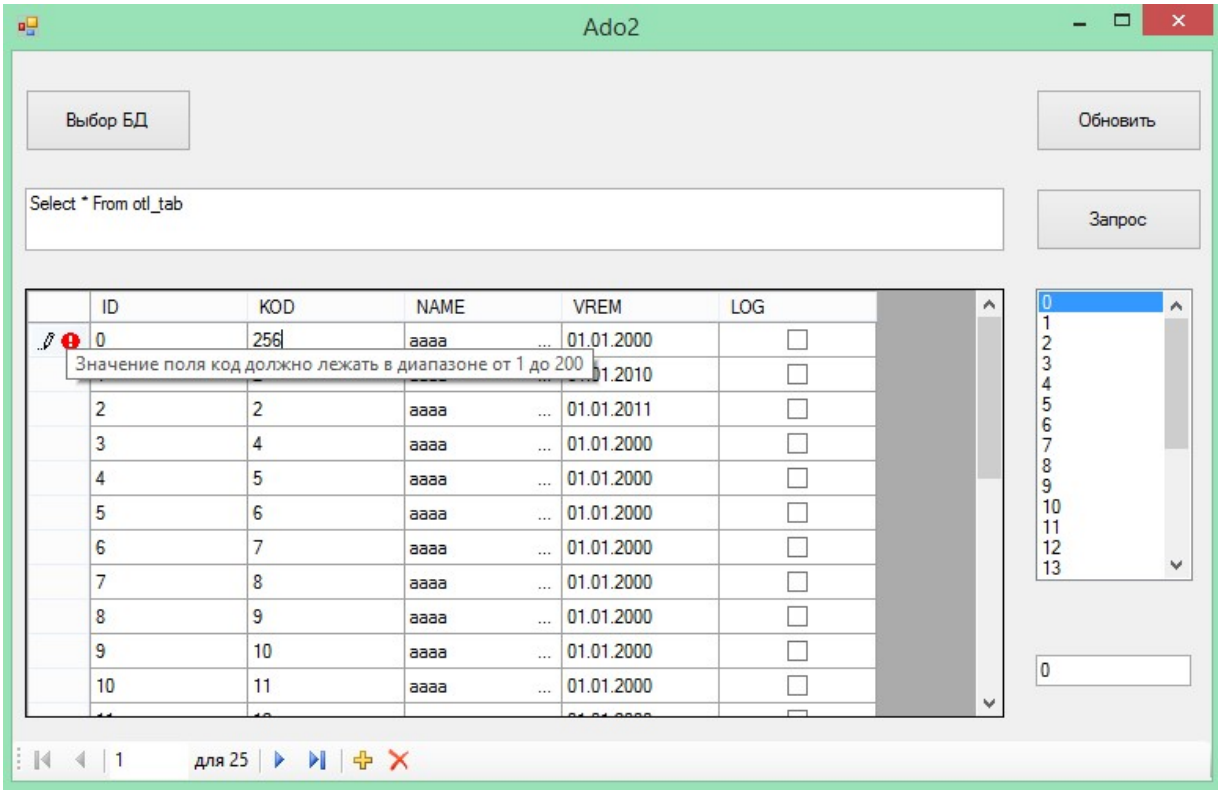


Рис. 7.6. Контроль введення даних

Після успішного введення даних слід стерти текст повідомлення помилки. Це робиться у випадку завершення редагування комірки **CellEndEdit**. Наведемо код цієї події:

```
 //Завершение редактирования ячейки
private void dataGridView1_CellEndEdit(object sender, DataGridViewCellEventArgs e)
 {
     dataGridView1.Rows[e.RowIndex].ErrorText = "";
 }
```

7.13. Приклад ADO_SortFilterFind

У цьому прикладі продемонструємо всі способи сортування, фільтрації та пошуку даних, які застосовуються в **ADO.NET**. Кожен із методів перевірявся на таблиці з одним мільйоном записів.

Для початку роботи в конструкторі форм **Designer** спроектуємо форму, де розмістимо необхідні елементи керування (рис. 7.7).

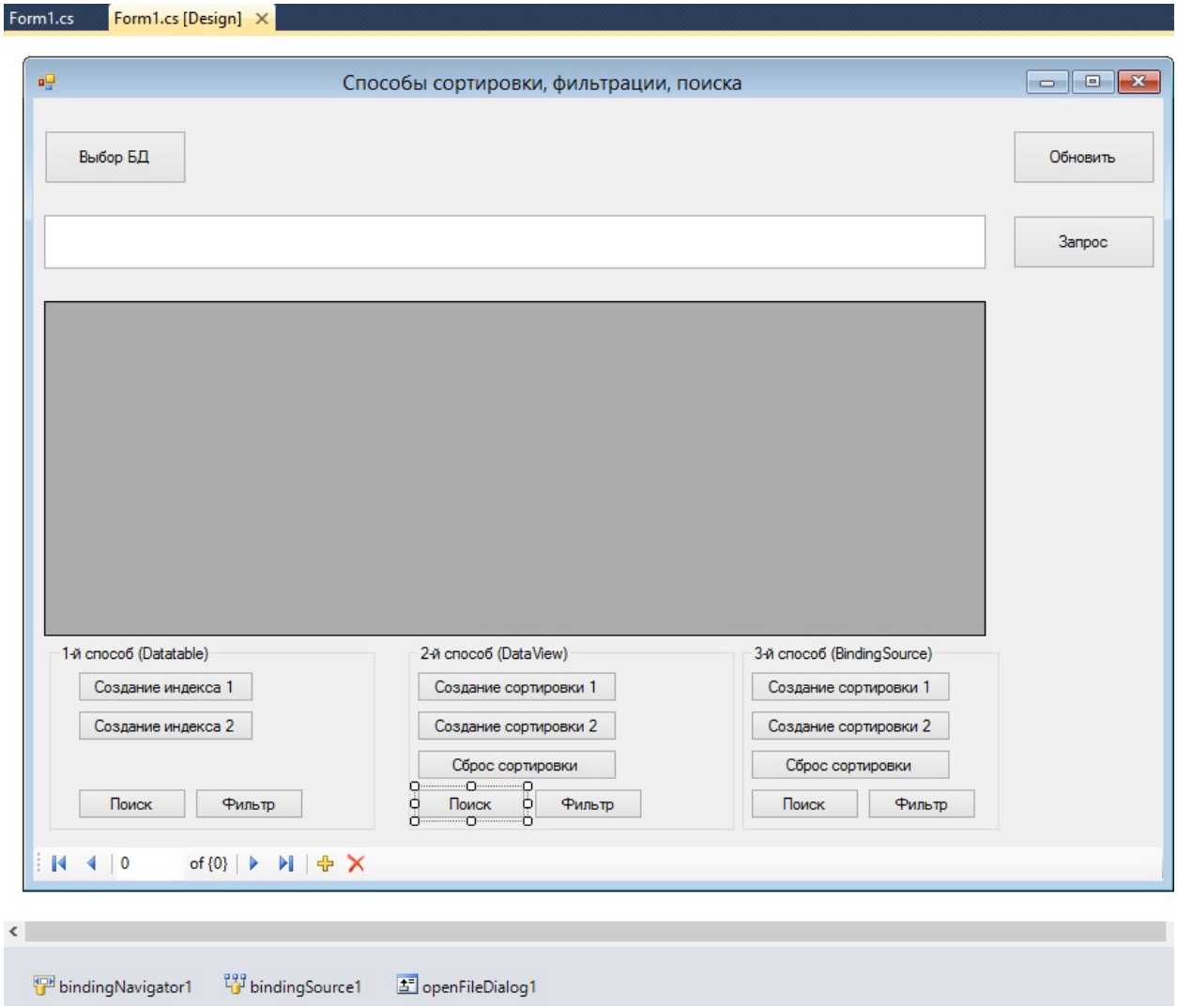


Рис. 7.7. Конструювання додатку ADO_SortFilterFind

Наведемо повний текст програмного коду програми ADO_SortFilterFind:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OleDb;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Ado22
{
    public partial class Form1 : Form
    {
        //Объект Connection с драйвером OleDb
        OleDbConnection conn;
        OleDbDataAdapter da;
        DataSet ds;

        public Form1()
        {
            InitializeComponent();
            conn = null;
            da = null;
            ds = new DataSet();
        }

        //Открытие БД
        private void button1_Click(object sender, EventArgs e)
        {
            //Открытие файла *.mdb
            openFileDialog1.FileName = "otl.mdb";
            openFileDialog1.Filter = "*.mdb|*.mdb";
            if (openFileDialog1.ShowDialog() != DialogResult.OK) return;

            //Формирование объекта Connection
            try
            {
                string source = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
                                openFileDialog1.FileName + ";Mode=ReadWrite";
                if (conn != null) conn.Dispose();
                conn = new OleDbConnection(source);
                conn.Open();
                textBox1.Text = "Select * From otl_tab";
            }

            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

        }
        //Получение результата SQL-запроса
        private void button2_Click(object sender, EventArgs e)
        {
            try
            {
                //Создание объекта Recordset
                if (da != null) da.Dispose();
                da = new OleDbDataAdapter(textBox1.Text, conn);

                //Сохранение данных через объект OleDbCommandBuilder
```

```
        OleDbCommandBuilder bulder = new OleDbCommandBuilder(da);
        //Символы в которые будут заключены поля и таблицы БД
        //при формировании запросов
        bulder.QuotePrefix = "[";
        bulder.QuoteSuffix = "]";

        //Сформированные запросы на Insert, Delete, Update
        string str = bulder.GetInsertCommand().CommandText+"\n\n";
        str = str + bulder.GetDeleteCommand().CommandText + "\n\n";
        str = str + bulder.GetUpdateCommand().CommandText;
        MessageBox.Show(str);

        if (ds.Tables.Count>0) ds.Tables.RemoveAt(0);
        da.Fill(ds);

        bindingSource1.DataSource = ds.Tables[0];
        dataGridView1.DataSource = bindingSource1;
        bindingNavigator1.BindingSource = bindingSource1;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Обновление данных
private void button3_Click(object sender, EventArgs e)
{
    try
    {
        da.Update(ds.Tables[0]);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Работа с индексами, сортировкой и фильтрацией с помощью объекта DataTable
////////////////////////////////////////////////////////////////////////////

//Primarykey - создание индекса (уникальное поле таблицы)
//Find - поиск точного значения по индексированному полю
//Select - фильтрация данных, работает как с индексом так и без него (с индексом
//фильтрация данных работает мгновенно)
//Данный способ является самым быстрым и оптимальным - он позволяет хранить все
//индексы и производить быстрый и эффективный поиск

//Создание индекса 1
private void button4_Click(object sender, EventArgs e)
{
    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str;
      //Создание индексного файла по 1-му полю таблицы
      //При создании следующих индексов-все предыдущие сохраняются.
      //При создании индекса PrimaryKey значения в столбце должны быть уникальны
        ds.Tables[0].PrimaryKey = new DataColumn[] { ds.Tables[0].Columns[0] };

        dt2 = DateTime.Now;
        tt = dt2 - dt1;
```

```
        str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
               1000.0).ToString() + " секунд";
        MessageBox.Show(str);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Создание индекса 2
private void button5_Click(object sender, EventArgs e)
{
    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str;

        //Создание индексного файла по 2-му полю таблицы
        ds.Tables[0].PrimaryKey = new DataColumn[] { ds.Tables[0].Columns[1] };
        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
              1000.0).ToString() + " секунд";
        MessageBox.Show(str);

    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Поиск с помощью метода Find в таблице с последующим переходом на значение
private void button7_Click(object sender, EventArgs e)
{

    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str="";

 //Поиск на точное совпадение - делается по индексному полю (Primarykey) таблицы
        //При существующем индексе делается мгновенно
        DataRow zap = ds.Tables[0].Rows.Find(textBox1.Text);
        DataView datav = ds.Tables[0].DefaultView;

        if (zap != null)
        {
            int pos = datav.Table.Rows.IndexOf(zap);
            dataGridView1.CurrentCell = dataGridView1.Rows[pos].Cells[0];

        }
        else
            str = "Значение не найдено\n";
```

```
        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = str + "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds
                    / 1000.0).ToString() + " секунд";
        MessageBox.Show(str);

    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Фильтрация данных с помощью метода Select в таблице
//Фильтр работает с индексными файлами и без них.
//Фильтр с учетом индекса делается мгновенно!!!
private void button8_Click(object sender, EventArgs e)
{
    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str="";

        DataRow [] datar = ds.Tables[0].Select(textBox1.Text);
        DataTable table = ds.Tables[0];

        DataTable newTable = new DataTable();
        DataRow vr;
        int i;

        for (i = 0; i < table.Columns.Count;i++)
    newTable.Columns.Add(table.Columns[i].ColumnName,table.Columns[i].DataType);

        foreach (DataRow row in datar)
        {

            vr = newTable.NewRow();

            for (i = 0; i < table.Columns.Count; i++)
                vr[i]=row[i];

            newTable.Rows.Add(vr);
        }

        //dataGridView1.DataSource = null;
        dataGridView1.DataSource = newTable;

        //dataGridView  1. DataSource = ;

        /*
        DataView datav = ds.Tables[0].DefaultView;
        if (datar.Length > 0)
        {
            int pos = datav.Table.Rows.IndexOf(datar[0]);
            dataGridView1.CurrentCell = dataGridView1.Rows[pos].Cells[0];
        }
        else
            str = "Нет значений, которые удовлетворяют фильтру\n";
        */
```

```
        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = str + "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds
                    / 1000.0).ToString() + " секунд";
        MessageBox.Show(str);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Работа сортировкой, поиском и фильтрацией с помощью объекта DataView
//Sort - сортировка по выбранному полю или совокупности полей(влияет на
// представление данных и в текущий момент времени активна только одна)
//Find - точный поиск, который осуществляется по выбранной сортировке (работает
//быстро)
//RowFilter - фильтрация данных (работает независимо от сортировки) - всегда
 //работает медленно

//Сортировка и фильтрация с помощью DataView предназначена для перестраивания
//представления данных, генерации отчетов и т.д.
//Для вычислительных операций эффективнее всего использование методов объекта
//DataTable

//Сортировка по 1-му полю
private void button9_Click(object sender, EventArgs e)
{
    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str;

        //Сортировка по первому полю
        DataView datav = ds.Tables[0].DefaultView;
        datav.Sort = ds.Tables[0].Columns[0].ColumnName;

        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
              1000.0).ToString() + " секунд";
        MessageBox.Show(str);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Сортировка по 2-му полю
private void button10_Click(object sender, EventArgs e)
{
    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
```

```
        ////////////////////////////////////////
        string str;

        //Сортировка по второму полю
        DataView datav = ds.Tables[0].DefaultView;
        datav.Sort = ds.Tables[0].Columns[1].ColumnName;

        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
               1000.0).ToString() + " секунд";
        MessageBox.Show(str);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Сброс сортировки
private void button11_Click(object sender, EventArgs e)
{
    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str;
        //Сброс сортировки
        DataView datav = ds.Tables[0].DefaultView;
        datav.Sort = "";

        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
               1000.0).ToString() + " секунд";
        MessageBox.Show(str);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Поиск с помощью метода Find по отсортированному полю с использованием DataView
private void button12_Click(object sender, EventArgs e)
{
    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str="";

        //Поиск с помощью метода Find по отсортированному полю
        DataView datav = ds.Tables[0].DefaultView;
        int pos = datav.Find(textBox1.Text);
        if (pos >= 0)
            dataGridView1.CurrentCell = dataGridView1.Rows[pos].Cells[0];
        else
            str = "Значение не найдено\n";
```

```
        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = str + "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds
                    / 1000.0).ToString() + " секунд";
        MessageBox.Show(str);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Фильтрация данных с помощью метода RowFilter - всегда выполняется медленно
private void button13_Click(object sender, EventArgs e)
{

    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str = "";

     //Фильтрация данных с помощью метода RowFilter - всегда выполняется медленно
        DataView datav = ds.Tables[0].DefaultView;
        datav.RowFilter = textBox1.Text;
        if (datav.Count < 1)
            str = "Нет значений, которые удовлетворяют фильтру\n";

        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = str + "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds
        / 1000.0).ToString() + " секунд";
        MessageBox.Show(str);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Работа сортировкой, поиском и фильтрацией с помощью объекта BindingSource
////////////////////////////////////////////////////////////////////

//Сортировка по 1-му полю
private void button14_Click(object sender, EventArgs e)
{
    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str;

        //Сортировка по первому полю
        bindingSource1.Sort = ds.Tables[0].Columns[0].ColumnName;

        dt2 = DateTime.Now;
        tt = dt2 - dt1;
```

```
        str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
               1000.0).ToString() + " секунд";
        MessageBox.Show(str);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Сортировка по 2-му полю
private void button15_Click(object sender, EventArgs e)
{
    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str;

        //Сортировка по второму полю
        bindingSource1.Sort = ds.Tables[0].Columns[1].ColumnName;

        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
               1000.0).ToString() + " секунд";
        MessageBox.Show(str);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Сброс сортировки
private void button17_Click(object sender, EventArgs e)
{
    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str;
        //Сброс сортировки
        bindingSource1.Sort = "";

        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
                1000.0).ToString() + " секунд";
        MessageBox.Show(str);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

```
        //Поиск с помощью метода Find  без учета сортировки (осуществляется
        //последовательный поиск - всегда медленно)
        private void button16_Click(object sender, EventArgs e)
        {
            try
            {
                DateTime dt1, dt2;
                TimeSpan tt;
                dt1 = DateTime.Now;
                ////////////////////////////////////////
                string str = "";

                //Поиск с помощью метода Find  без учета сортировки (осуществляется
                //последовательный поиск - всегда медленно)
                int pos = bindingSource1.Find("id",textBox1.Text);
                if (pos >= 0)
                    dataGridView1.CurrentCell = dataGridView1.Rows[pos].Cells[0];
                else
                    str = "Значение не найдено\n";

                dt2 = DateTime.Now;
                tt = dt2 - dt1;

                str = str + "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds
                      / 1000.0).ToString() + " секунд";
                MessageBox.Show(str);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        //Фильтрация данных с помощью метода Filter - всегда выполняется медленно
        private void button18_Click(object sender, EventArgs e)
        {
            try
            {
                DateTime dt1, dt2;
                TimeSpan tt;
                dt1 = DateTime.Now;
                ////////////////////////////////////////
                string str = "";

                //Фильтрация данных с помощью метода Filter - всегда выполняется медленно
                bindingSource1.Filter = textBox1.Text;
                DataView datav = ds.Tables[0].DefaultView;
                if (datav.Count < 1)
                    str = "Нет значений, которые удовлетворяют фильтру\n";

                dt2 = DateTime.Now;
                tt = dt2 - dt1;

                str = str + "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds
                / 1000.0).ToString() + " секунд";
                MessageBox.Show(str);
            }

            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

    }
}
```

Існує три способи роботи з сортуванням, фільтрацією та пошуком даних:

1. Використовуючи методи класу **DataTable**.
2. Використовуючи методи класу представлення **DataView**.
3. Використовуючи методи зв'язувача **BindingSource**.

При першому способі користувачеві необхідно створити індекси для полів, де відбуватиметься пошук або фільтрація даних. Можна створювати індекси, які складаються з кількох полів. Поле, за яким створюється індекс, має бути унікальним (властивість **PrimaryKey**). Як **PrimaryKey** необхідно вказати масив колонок (**DataColumn**).

Приклад створення індексу першого поля таблиці:

```
ds.Tables[0].PrimaryKey = new DataColumn[] { ds.Tables[0].Columns[0] };
```

Важливим є те, що всі індекси таблиці зберігаються, але створюються при початковому присвоєння. Так, індекси створюються лише один раз, а потім просто перемикаються.

Пошук здійснюється методом **Find** властивості таблиці **Rows** та повертає рядок типу **DataRow**. Якщо пошук невдалий – значення **null**. Під час пошуку вказується значення вибраного ключового поля. Метод **Find** шукає точне значення за активним ключем (властивість **PrimaryKey**).

Приклад пошуку:

```
DataRow zap = ds.Tables[0].Rows.Find(500000);
```

Для фільтрації даних використовується функція **Select**, яка повертає масив рядків **DataRow[]**, що задовольняють заданій умові. Фільтрування зі створеними індексними файлами працює миттєво. При цьому не має значення, який активний індекс обрано. Фільтр може працювати і без індексів, але це буде неефективно і займе набагато більше часу.

Приклад фільтрації даних:

```
DataRow [] datar = ds.Tables[0].Select("id>500000 and id<700000");
```

Висновок

Для рішень різноманітних обчислювальних завдань цей спосіб є найефективнішим. Користувач один раз створює необхідні індекси та робить пошук та фільтрацію з максимальною швидкістю.

Другий і третій способи працюють з представленням (об'єкт **DataView**), яке має відображатися в елементах керування, наприклад, **DataGridView**. Це зручно для організації звітів, де потрібно візуалізувати (представити у наочному вигляді) дані.

Об'єкт **DataView** містить такі властивості та методи:

1. **Sort** – властивість, яка задає сортування і перебудовує об’єкт представлення (сортування може бути задане по кількох полях). У даний момент активне тільки одне сортування.

Приклад сортування:

```
DataView datav = ds.Tables[0].DefaultView;
datav.Sort = ds.Tables[0].Columns[0].ColumnName;
```

2. **Find** – метод точного пошуку за ключем сортування (Sort) – може містити кілька полів для пошуку. **Без встановлення сортування Find не працюватиме!** При успішному пошуку повертається позиція знайденого рядка представлення, інакше – значення -1.

Приклад пошуку:

```
DataView datav = ds.Tables[0].DefaultView;
int pos = datav.Find(textBox1.Text);
```

3. **RowFilter** – метод фільтрації даних. На великих обсягах даних цей метод працює дуже довго, незважаючи на встановлену властивість **Sort**. Не виявлено особливих відмінностей при роботі методу з сортуванням та без нього.

Приклад фільтрації даних:

```
DataView datav = ds.Tables[0].DefaultView;
datav.RowFilter = "id>500000 and id<700000";
```

При роботі з елементом **BindingSource** спостерігаємо аналогічні результати сортування та фільтрації даних. Щодо пошуку, то він робиться послідовно і без урахування сортування. Такий пошук застосовувати недоцільно.

Приклад сортування через **BindingSource**:

```
bindingSource1.Sort = ds.Tables[0].Columns[0].ColumnName;
```

Пиклад фільтрації за допомогою **BindingSource**:

```
bindingSource1.Filter = "id>500000 and id<700000";
```

Приклад пошуку за допомогою **BindingSource**:

```
int pos = bindingSource1.Find("id",500000);
```

Загальний висновок:

Для ефективного пошуку та фільтрації даних, а також створення відразу кількох індексів слід користуватися методами та властивостями класу

DataTable (**PrimaryKey** – створення та вибір індексу, **Find** – пошук, **Select** – фільтрація). При цьому і фільтр, і пошук працюватимуть миттєво.

Для зміни представлення даних в елементах керування, скажімо для формування звітів, можна використовувати методи класу **DataView** (**Sort**, **Find**, **RowFilter**) або **BindingSource** зв'язувача (**Sort**, **Find**, **Filter**). При цьому буде не висока швидкість на великих обсягах даних, оскільки головна мета сортування та фільтрації перебудувати представлення існуючої таблиці для відображення у відповідному елементі.

На рис. 7.8. наведемо результат виконання програми ADO_SortFilterFind.

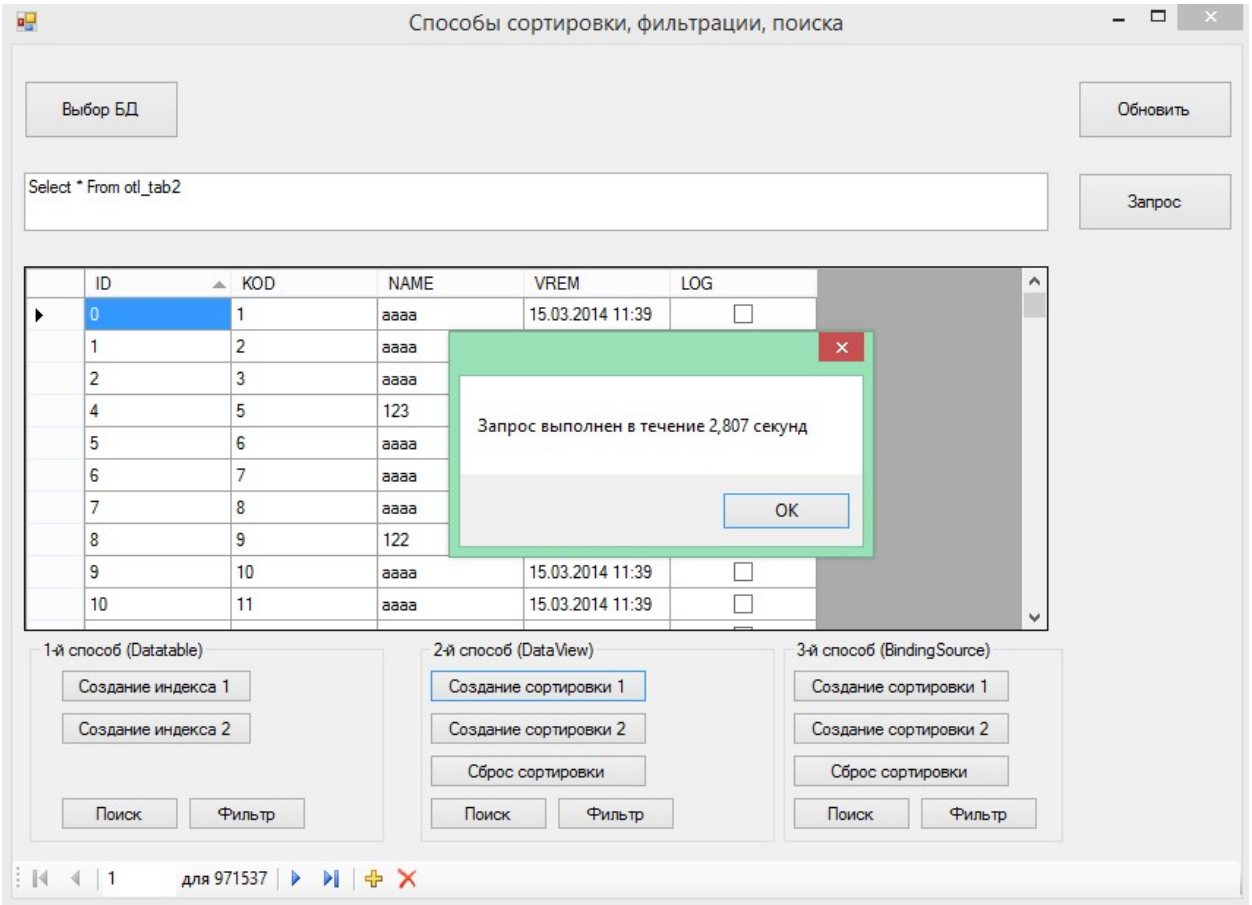


Рис. 7.8. Результат роботи програми ADO_SortFilterFind

7.14. Приклад AdoRelation1

Працюючи з технологією **ADO.NET** об'єкт **DataSet** є сукупність таблиць моделі бази даних на боці клієнта. У цій моделі можна здійснювати реляційні зв'язки між таблицями за допомогою властивості **Relations** об'єкта **DataSet**, який повертає **DataRelation** об'єкт. Оскільки в елементі керування **DataGridView** можна розмістити лише одну таблицю, для демонстрації зв'язку кількох таблиць розмістимо два об'єкти класу **DataGridView**, відповідно **dataGridView1** та **dataGridView2**.

Як приклад реляційних зв'язків наведемо зв'язок «один-до-багатьох» на прикладі таблиць бази даних «nwind.mdb» **Categories** і **Products**. Так, одній категорії товарів відповідає багато різних продуктів.

Наведемо повний текст програмного коду **ADORelation1**:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OleDb;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Ado1
{
    public partial class Form1 : Form
    {
        //Объект Connection
        OleDbConnection conn;
        DataSet ds;

        public Form1()
        {
            InitializeComponent();

            //Создание объекта Connection и Recordset
            try
            {
                string source = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=NWIND.MDB;Mode=ReadWrite";

                string select = "SELECT CategoryID,CategoryName FROM Categories";
                conn = new OleDbConnection(source);
                conn.Open();

                ds = new DataSet();
                OleDbDataAdapter da1 = new OleDbDataAdapter(select, conn);

                da1.Fill(ds,"Категория");
                da1.TableMappings.Add("Categories", "Категория");

                select = "SELECT CategoryID,ProductName FROM Products";
                OleDbDataAdapter da2 = new OleDbDataAdapter(select, conn);

                da2.Fill(ds,"Продукты");
                da1.TableMappings.Add("Products", "Продукты");

                ds.Relations.Add("Продукты категории", ds.Tables[0].Columns[0],
                                 ds.Tables[1].Columns[0]);
                dataGridView1.DataSource = ds;
                dataGridView1.DataMember = "Категория";

                dataGridView2.DataSource = ds;
                dataGridView2.DataMember = "Категория.Продукты категории";
            }
```

```
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

У цьому прикладі в одному об’єкті **DataSet** створюємо дві таблиці – об’єкти **DataTable**. За допомогою властивості **TableMappings** об’єкта **DataAdapter** створюємо псевдонім для таблиць. Для таблиці "Categories" виберемо псевдонім "Категорія", для таблиці "Products" - "Продукти". Далі при побудові запитів можна скористатися даними псевдонімами.

Далі створимо реляційні зв’язки між таблицями, використовуючи властивість **Relations** об’єкта **DataSet**. Зв’язок «один-до-багатьох» здійснюється за ключовим полем «CategoryID»:

```
.....................................................................
            ds.Relations.Add("Продукты категории", ds.Tables[0].Columns[0],
                             ds.Tables[1].Columns[0]);
.....................................................................
```

Ім’я реляційного зв’язку - "Продукти категорії".

На наступному етапі прив’яжемо таблицю "Категорія" з первинним ключем до елемента ***dataGridView1***.

```
.....................................................................
            dataGridView1.DataSource = ds;
            dataGridView1.DataMember = "Категория";
.....................................................................
```

А таблицю із зовнішнім ключем – до елемента **dataGridView2**. При цьому зазначимо реляційний зв’язок «Категорія. Продукти категорії».

```
.....................................................................
            dataGridView2.DataSource = ds;
            dataGridView2.DataMember = "Категория.Продукты категории";
.....................................................................
```

Такий запис означає, що при переміщенні на певний запис у головній таблиці «Категорія» – у другій таблиці ми бачитимемо лише ті товари, які відповідають обраній категорії.

На рис. 7.9. наведемо результат виконання програми **ADORelation1**.

Так, у другому елементі **dataGridView2** ми бачимо лише продукти обраної 6-ї категорії (поле **CategoryID**).

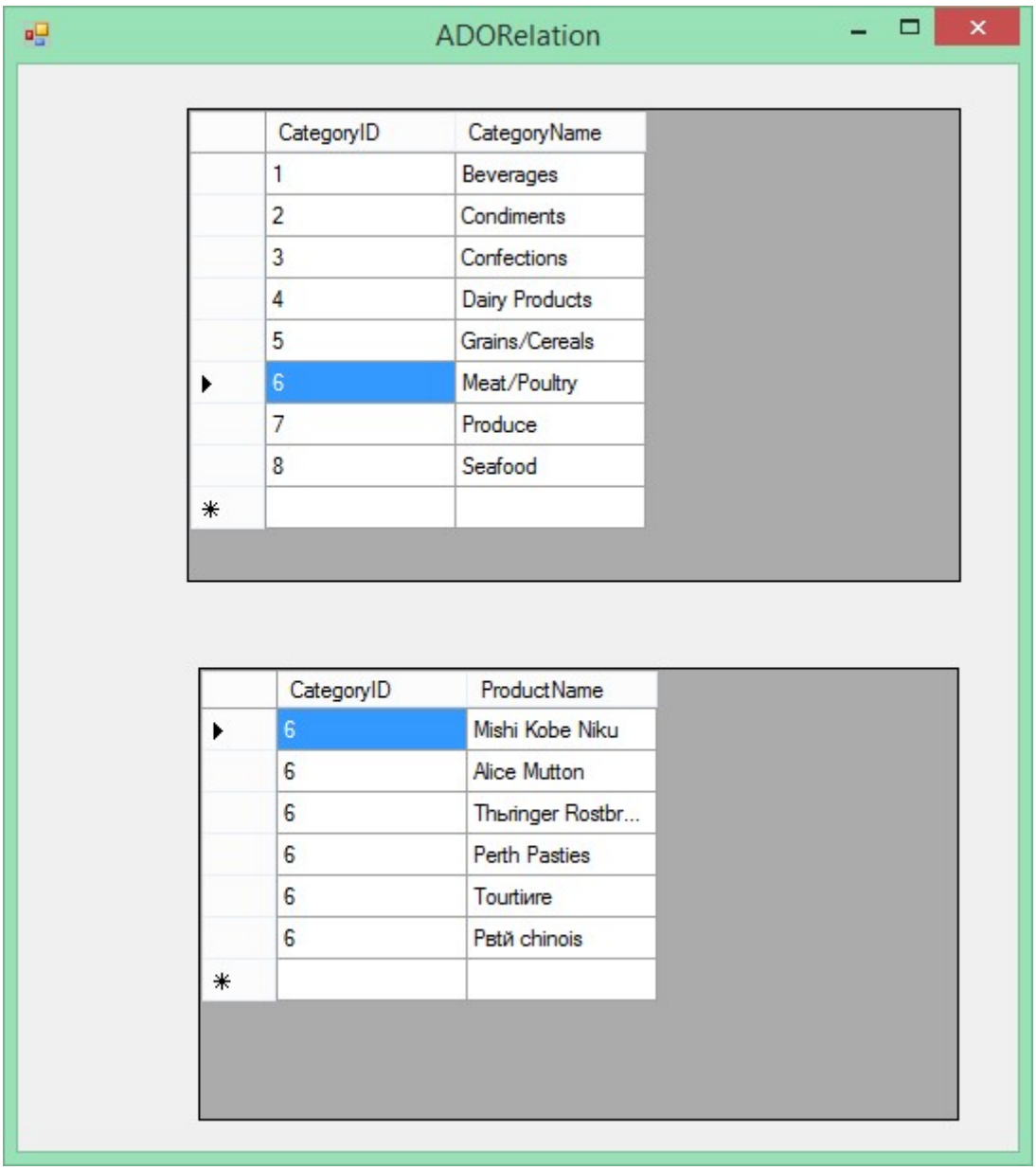


Рис. 7.9. Результат роботи програми ADORelation1

7.15. Приклад AdoRelation2

У прикладі **ADORelation2** також здійснюється реляційний зв'язок «один-до-багатьох». Відмінністю є застосування старішого елемента керування відображення таблиць **DataGrid**. Його перевагою є відображення підлеглих таблиць при зв'язку одним так, як це реалізовано в **MS Access**.

Наведемо повний текст програмного коду **ADORelation2**:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OleDb;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
```

```
using System.Windows.Forms;

namespace Ado1
{
    public partial class Form1 : Form
    {
        //Объект Connection
        OleDbConnection conn;
        DataSet ds;

        public Form1()
        {
            InitializeComponent();

            //Создание объекта Connection и Recordset
            try
            {
                string source = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=NWIND.MDB;Mode=ReadWrite";

                string select = "SELECT CategoryID,CategoryName FROM Categories";
                conn = new OleDbConnection(source);
                conn.Open();

                ds = new DataSet();

                OleDbDataAdapter da1 = new OleDbDataAdapter(select, conn);

                da1.Fill(ds,"Категория");
                da1.TableMappings.Add("Categories", "Категория");

                select = "SELECT CategoryID,ProductName FROM Products";
                OleDbDataAdapter da2 = new OleDbDataAdapter(select, conn);

                da2.Fill(ds,"Продукты");
                da1.TableMappings.Add("Products", "Продукты");

                ds.Relations.Add("Продукты категории", ds.Tables[0].Columns[0],
                                 ds.Tables[1].Columns[0]);

                dataGridView1.DataSource = ds.Tables[0];
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

На рис. 7.10. наведемо результат виконання програми **ADORelation2**.

На відміну від прикладу **AdoRelation1** дані батьківської та підлеглої таблиці можна побачити в елементі керування **DataGrid**. Для перегляду підлеглих даних необхідно натиснути "+".

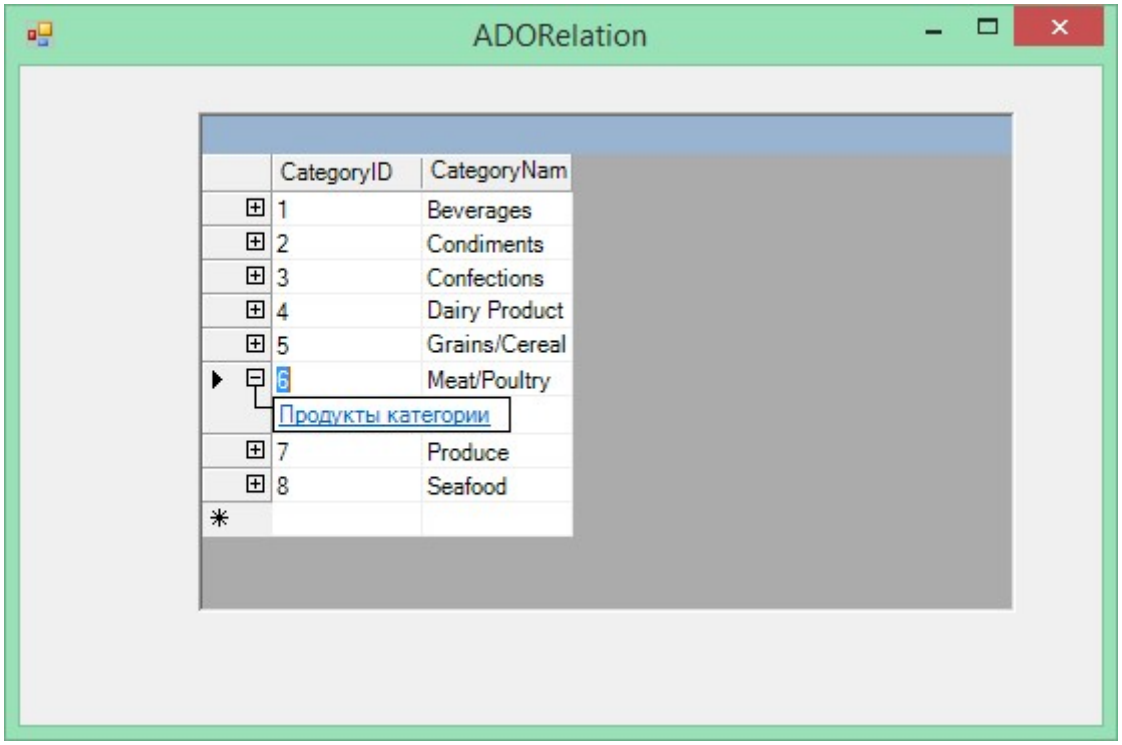


Рис. 7.10. Результат роботи програми ADORelation2

7.16. Приклад ADO_MDB

Приклад **ADO_MDB** максимально відображає всі особливості технології **ADO.NET**.

Робота прикладу **ADO_MDB** призначена для роботи з базами даних **Microsoft ACCESS** (***.mdb**) через **OLE DB**. Для демонстрації повної функціональності технології **ADO. NET** у якості бази даних необхідно вибрати **otl.mdb**. Дані БД **otl.mdb** мають таблицю **otl_tab**. Структура даної таблиці наведена нижче (табл. 7.19)

Таблиця 7.19

Структура таблиці otl_tab

№ з/п	Им'я поля	Тип поля	Довжина
1	id	Числовий (довге ціле)	
2	kod	Числовий (довге ціле)	
3	Name	Текстовий	50
4	Vrem	Дата/час	
5	log	Логічний	

Для початку роботи в конструкторі форм **Designer** спроектуємо форму, де розмістимо необхідні елементи керування (рис. 7.11).

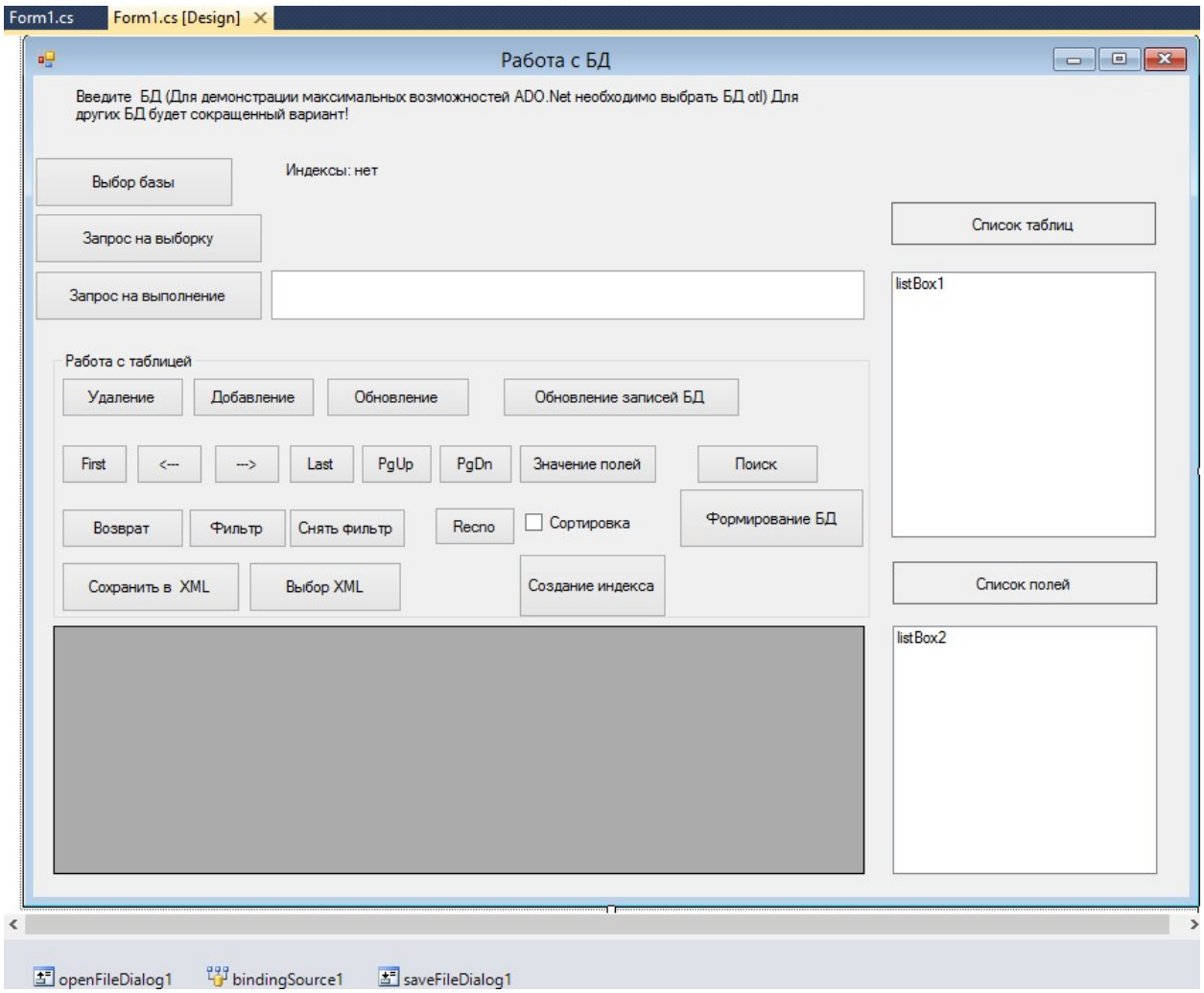


Рис. 7.11. Конструювання додатку ADO_MDB

Наведемо список об’єктів із їх функціональним призначенням:

1. Список listBox1 – список відображення таблиць обраної БД.
2. Список listBox2 – список для відображення полів вибраної таблиці.
3. Текстове поле textBox1 призначене для введення запиту SQL.
4. Елемент openFileDialog1 призначений для вибору БД, а також завантаження файлу XML.
5. Елемент saveFileDialog1 призначений для збереження даних у файлі XML.
6. Елемент bindingSource1 – універсальний зв’язувач для прив’язування елементів до вибраної таблиці.
7. Елемент dataGridView1 – призначений для роботи з набором вибраної бази даних.
8. Кнопка «Вибір бази» призначена для вибору бази даних *.mdb та формування об’єкта Connection.
9. Кнопка «Запит на вибірку» – реалізація SQL-запитів на вибірку, записаних у textBox1.
10. Кнопка «Запит виконання» – реалізація SQL-запитів виконання (INSERT INTO, DELETE, UPDATE, DROP TABLE, CREATE TABLE тощо.) записаних у textBox1.

11. Кнопка «Видалення» – видалення поточного запису.
12. Кнопка «Додавання» – додавання нового запису.
13. Кнопка «Оновлення» – оновлення поточного запису.
14. Кнопка «Оновлення записів БД» - оновлення даних БД на стороні сервера.
15. Кнопка «First» – перехід на перший запис.
16. Кнопка «←» – перехід до попереднього запису.
17. Кнопка «→» – перехід до наступного запису.
18. Кнопка «Last» – перехід на останній запис.
19. Кнопка «PgUp» – перехід на 12 записів вгору.
20. Кнопка «PgDn» – перехід на 12 записів вниз.
21. Кнопка «Значення полів» – відображення значень полів на екрані.
22. Кнопка «Пошук» – пошук даних за заданою умовою.
23. Кнопка «Повернення» - повернення даних таблиці у вихідний стан.
24. Кнопка «Фільтр» – встановлення фільтру за заданою умовою.
25. Кнопка «Зняти фільтр» – зняття фільтра.
26. Кнопка «Recno» – повернення абсолютної позиції запису.
27. Прапорець «Сортування» – встановлення та зняття сортування за певним полем таблиці.
28. Кнопка «Формування БД» – формування даних таблиці "otl_tab" бази даних "otl.mdb".
29. Кнопка «Зберегти у XML» – збереження структури та даних вибраної таблиці у файл *.xml.
30. Кнопка «Вибір XML» – читання структури та даних із файлу *.xml. таблицю (об’єкт DataTable).
31. Кнопка «Створення індексу» призначена для створення індексних файлів за вибраними полями.

Так як текст програми дуже об’ємний, доцільно навести функції відгуків кнопок, реалізованих у цьому прикладі окремо з їх функціональними особливостями та детальним описом.

Наведемо повний текст програмного коду ADO_MDB і зупинимося на найважливіших моментах.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OleDb;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Ado3
{
    public partial class Form1 : Form
    {

        //Объект Connection с драйвером OleDb
```

```
        OleDbConnection conn;
        OleDbDataAdapter da;
        OleDbCommand cmd;
        DataSet ds;

        //Переменные текущей БД и выбранной таблицы
        string baza;
        string table;

        //Переменная количества индексных файлов
        int kol_index = 0;

        public Form1()
        {
            InitializeComponent();

            conn = null;
            da = null;
            cmd = null;
            ds = null;
            //Строка для изменения масштаба приложения
            //Scale(0.8f, 0.8f);
            show(false);
        }

        //----------------------Функции для работы с базой данных-----------------------
-

     //Функция отображения/скрытия элементов управления
     //kod_bd = 0 - полный доступ
     //kod_bd = 1 - для таблиц отличных от otl_tab
     //kod_bd = 2 - для фильтра

     private void show (bool kod=true,int kod_bd = 0)
    {
        dataGridView1.Visible = kod;
        textBox1.Visible = kod;
        Zapros_Select.Visible = kod;
        Filter.Visible = kod;
        Filter_off.Visible = kod;
        ButtonFirst.Visible = kod;
        ButtonLast.Visible = kod;
        ButtonLeft.Visible = kod;
        ButtonRight.Visible = kod;
        ButtonPgdn.Visible = kod;
        ButtonPgup.Visible = kod;
        ButtonRecno.Visible = kod;

        label1.Visible = kod;
        label2.Visible = kod;
        label3.Visible = kod;
        label4.Visible = kod;
        listBox1.Visible = kod;
        listBox2.Visible = kod;
        groupBox1.Visible = kod;

        if (kod_bd == 2) kod = false;

        Zapros_Make.Visible = kod;
        Zapis_Delete.Visible = kod;
        Find.Visible = kod;
        Vozvrat.Visible = kod;
        Check_Sort.Visible = kod;
        CreateIndex.Visible = kod;
        ButtonSaveXML.Visible = kod;
```

```
    ButtonLoadXML.Visible = kod;
    Update_BD.Visible = kod;
if(kod_bd>0)kod = false;

    Zapis_Add.Visible = kod;
    Zapis_Update.Visible = kod;
    Values_Fields.Visible = kod;
    Formir_BD.Visible = kod;
}

    //Отлик на кнопку Выбор БД
    private void Vibor_BD_Click(object sender, EventArgs e)
    {
        //Открытие файла *.mdb
        openFileDialog1.FileName = "otl.mdb";
        openFileDialog1.Filter = "*.mdb|*.mdb";
        if (openFileDialog1.ShowDialog() != DialogResult.OK) return;

        //Формирование объекта Connection

        dataGridView1.DataSource = null;
        try
        {
            string source = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
                openFileDialog1.FileName + ";Mode=ReadWrite";

            if (conn != null && conn.State == ConnectionState.Open) conn.Close();
            if (conn != null) conn.Dispose();
            conn = new OleDbConnection(source);
            label1.Text = openFileDialog1.FileName;

            baza = openFileDialog1.SafeFileName.ToUpper();

            //MessageBox.Show(source, "Начальная строка подключения",
            //MessageBoxButtons.OK, MessageBoxIcon.Information);

            conn.Open();

            show(true, 1);

            if (ds != null) ds.Dispose();
            ds = new DataSet();
            Structura_BD();
            textBox1.Text = "";

            kol_index = 0;
            label4.Text = "Индексы : нет";
        }

        catch (Exception ex)
        {
            if (conn != null && conn.State == ConnectionState.Open) conn.Close();
            if (conn != null) conn.Dispose();
            conn = null;

            if (ds != null) ds.Dispose();
            ds = null;
            show(false);
            MessageBox.Show(ex.Message);
        }
    }

    // Вывод структуры БД (список таблиц БД и полей по выбранной таблице)
    void Structura_BD()
    {
```

```
        DataTable table = conn.GetSchema("Tables");
        //Формирование списка таблиц с типом таблицы TABLE

        listBox1.Items.Clear();

        foreach (DataRow row in table.Rows)
        {
            if (row["TABLE_TYPE"].ToString() == "TABLE")
                listBox1.Items.Add(row["TABLE_NAME"]);
        }
    }

//---------------------------Функции откликов-------------------------------------

    // Вывод полей в листбокс по заданной таблице
    private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        string str;
        try
        {
            table = listBox1.SelectedItem.ToString();
            str = "Select * From [" + table + "]";

            if (da != null) da.Dispose();

            da = new OleDbDataAdapter(str, conn);
            ds.Tables.Clear();

            da.Fill(ds);

            listBox2.Items.Clear();
            DataTable table1 = ds.Tables[0];

            for (int i = 0; i < table1.Columns.Count; i++)
                listBox2.Items.Add(table1.Columns[i].ColumnName + "  -  " +
                    table1.Columns[i].DataType);
        }

        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            return;
        }

        textBox1.Text = str;
        Zapros_Select_Click(null, EventArgs.Empty);
        groupBox1.Text = "Работа с таблицей " + table;
    }


    //Запрос на выборку
    private void Zapros_Select_Click(object sender, EventArgs e)
    {
        string str = textBox1.Text.ToUpper();

        if (str.Length<6||str.Substring(0, 6) != "SELECT")
        {
     MessageBox.Show("В запросе на выборку отсутствует параметр SELECT", "Ошибка!!!",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);
            textBox1.Text = "SELECT <*>|<список полей> FROM <список таблиц>";
            textBox1.Focus();
            return;
        }
```

```
    try
    {
        //Создание объекта Recordset
        if (da != null) da.Dispose();
        da = new OleDbDataAdapter(textBox1.Text, conn);
        ds.Tables.Clear();

        //Сохранение данных через объект OleDbCommandBuilder
        // Предусмотрено для удаления, обновления и добавления записей в БД
        // Наличие ключевого поля обязательно ! Учитывается уникальность записей.

        OleDbCommandBuilder bulder = new OleDbCommandBuilder(da);
        //Символы в которые будут заключены поля и таблицы БД
        //при формировании запросов
        bulder.QuotePrefix = "[";
        bulder.QuoteSuffix = "]";

        da.Fill(ds);

        //Привязка bindingSource к объекту DataTable
        //и dataGridView к объекту bindingSource

       bindingSource1.DataSource = ds.Tables[0];
       dataGridView1.DataSource = bindingSource1;

       //Показать элементы управления в зависимости от выбранной таблицы
   if (baza.Substring(0,3).ToUpper()=="OTL"&&table.ToUpper() == "OTL_TAB")show();
   else show(true, 1);

        label4.Text = "Индексы : нет";
        kol_index = 0;
        //Разрешить удаление, добавление и редактирование записей
        dataGridView1.AllowUserToAddRows = true;
        dataGridView1.AllowUserToDeleteRows = true;
        dataGridView1.EditMode = DataGridViewEditMode.EditOnKeystrokeOrF2;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

// Запрос на выполнение
private void Zapros_Make_Click(object sender, EventArgs e)
{

    //Проверка времени выполнения запроса
    DateTime dt1, dt2;
    TimeSpan tt;
    dt1 = DateTime.Now;
    ////////////////////////////////////////
    string str = textBox1.Text.ToUpper().Trim();
    string str1="";
    int kod = 0;
    label1.Text = "";

    if (str.Length > 2 && str.Substring(0, 3) == "SEL")
      { MessageBox.Show("Команда SELECT недопустима для данной кнопки"); return;}

    if (str.Length > 2 && str.Substring(0, 3) == "DEL")
    {
        str1 = "DELETE FROM <таблица> [ where<условие> ]";
        kod = 1;
    }
```

```
            if (str.Length > 2 && str.Substring(0, 3) == "UPD")
            {
                str1 = "UPDATE <таблица> SET <поле>=<выражение>, [< поле>=<выражение>]...
[where<условие>]";
                kod = 2;
            }

            if (str.Length > 2 && str.Substring(0, 3) == "INS")
            {
                str1 = "INSERT INTO <таблица> (<список имен полей>) VALUES (<список
значений>)";
                kod = 3;
            }

            if (str.Length > 2 && str.Substring(0, 3) == "CRE")
            {
              str1 = "CREATE TABLE <таблица> (<имя поля1> <тип>, <имя поля2> <тип>... )";
              kod = 4;
            }

            if (str.Length > 2 && str.Substring(0, 3) == "DRO")
            {
                str1 = "DROP TABLE <таблица>";
                kod = 5;
            }

            if (kod>0) label1.Text = "Синтаксис: " + str1;

            try
            {
                if (cmd != null) cmd.Dispose();
                //Работа с транзакциями через объект OleDbCommand
                cmd = new OleDbCommand(textBox1.Text, conn);
                cmd.Transaction = conn.BeginTransaction();
                   //Выполнить без получения запроса
                   cmd.ExecuteNonQuery();
                cmd.Transaction.Commit();
            }

            catch (Exception ex)
            {
                //Откат транзакции
                cmd.Transaction.Rollback();
                MessageBox.Show(ex.Message);

                if (kod > 0) textBox1.Text = str1;
                textBox1.Focus();
                return;
            }

            //Изменить запрос (перечитать набор)
            ds.Tables.Clear();

            da.Fill(ds);

            bindingSource1.DataSource = ds.Tables[0];
            dataGridView1.DataSource = bindingSource1;

            label4.Text = "Индексы: нет";
            kol_index = 0;

            ////////////////////////////////////////
            dt2 = DateTime.Now;
            tt = dt2 - dt1;
```

```
    str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
          1000.0).ToString() + " секунд";
    MessageBox.Show(str);

    textBox1.Text = "";
    label1.Text = "";
}

// Удаление текущей записи из датагрида
private void Zapis_Delete_Click(object sender, EventArgs e)
{

    try
    {

        DataRowView dr = (DataRowView) bindingSource1.Current;
        dr.Delete();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

// Добавление записей в таблицу otl_tab
private void Zapis_Add_Click(object sender, EventArgs e)
{
    try
    {
        //Формирование новой строки
        DataRow ins = ds.Tables[0].NewRow();

        ins[0] = 12345;
        ins[1] = 56789;
        ins[2] = "Привет12345";
        ins[3] = DateTime.Now;
        ins[4] = true;

        //Добавление новой строки в таблицу DataSet
        ds.Tables[0].Rows.Add(ins);

        //Переход на нужную ячейку в dataGridView1

        dataGridView1.CurrentCell =
        dataGridView1.Rows[ds.Tables[0].Rows.IndexOf(ins)].Cells[0];

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

// Обновление текущей записи
private void Zapis_Update_Click(object sender, EventArgs e)
{
    try
    {
        DataRow zap = ((DataRowView)(bindingSource1.Current)).Row;
        zap[0] = 54321;
        zap[1] = 98765;
        zap[2] = "Привет98765";
        zap[3] = DateTime.Now;
        zap[4] = false;
    }
```

```
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Обновление записей базы данных (до этого все изменения происходят только в
//DataGridView)
private void Update_BD_Click(object sender, EventArgs e)
{
    try
    {
        da.Update(ds.Tables[0]);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Переход на первую запись
private void ButtonFirst_Click(object sender, EventArgs e)
{
    try
    {

        bindingSource1.MoveFirst();
        //dataGridView1.CurrentCell = dataGridView1.Rows[0].Cells[0];
        //bindingSource1.Position = 0; 2- й вариант

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Переход на предыдущую запись
private void ButtonLeft_Click(object sender, EventArgs e)
{
    try
    {
        if (bindingSource1.Position == 0) return;
        bindingSource1.MovePrevious();
        dataGridView1.CurrentCell =
             dataGridView1.Rows[bindingSource1.Position].Cells[0];
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Переход на следующую запись
private void ButtonRight_Click(object sender, EventArgs e)
{
    try
    {
        if (bindingSource1.Position == bindingSource1.Count - 1) return;
        bindingSource1.MoveNext();
        dataGridView1.CurrentCell =
               dataGridView1.Rows[bindingSource1.Position].Cells[0];
    }
    catch (Exception ex)
```

```
    {
        MessageBox.Show(ex.Message);
    }
}

//Переход на последнюю запись
private void ButtonLast_Click(object sender, EventArgs e)
{
    try
    {
        bindingSource1.MoveLast();
        dataGridView1.CurrentCell =
               dataGridView1.Rows[bindingSource1.Position].Cells[0];
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Постраничный переход на 12 записей вверх
private void ButtonPgup_Click(object sender, EventArgs e)
{
    try
    {
        int idx = bindingSource1.Position - 12;
        idx = (idx<0)?0:idx;
        dataGridView1.CurrentCell = dataGridView1.Rows[idx].Cells[0];
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Постраничный переход на 12 записей вниз
private void ButtonPgdn_Click(object sender, EventArgs e)
{
    try
    {
        int idx = bindingSource1.Position + 12;
        idx = (idx > bindingSource1.Count - 1) ? bindingSource1.Count - 1 : idx;
        dataGridView1.CurrentCell = dataGridView1.Rows[idx].Cells[0];
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Вывод значений полей и расчетные операции по представлению данных таблицы

private void Values_Fields_Click(object sender, EventArgs e)
{
    try
    {
        //Получение объекта представления данных таблицы - то что мы видим
        DataView dv = ds.Tables[0].DefaultView;

        if (dv.Count == 0)
        {
            MessageBox.Show("Пустая таблица!!!", "Внимание!!!",
                            MessageBoxButtons.OK,MessageBoxIcon.Information);
            return;
        }
```

```
      //  MessageBox.Show(kol_zap.ToString());

        //Взять коллекцию столбцов
        DataColumnCollection cols = ds.Tables[0].Columns;
        DataRow zap = ((DataRowView)(bindingSource1.Current)).Row;
        int i;
        string str = "";

        for (i = 0; i < cols.Count; i++)
        {
            //Если поля NULL ToString() возвращает ""
            str += zap[i] + "  ";
        }

        MessageBox.Show(str);

        // Вычислительные операции по первой текущей записи
        int d = 0;
        bool per = !(Convert.IsDBNull(dv[0][0]) || Convert.IsDBNull(dv[0][1]));
        if (per)d = (int)dv[0][0] + (int)dv[0][1];
        str = string.Format("Сумма первых двух полей = {0}", d);
        MessageBox.Show(str);

        // Вычисление среднего арифметического - Вариант 1
        //dataGridView1.DataSource = null;// Отвязка от datagridView - при работе
        //можно не отвязываться

        int sum = 0, kol = 0;
        for (i = 0; i < dv.Count; i++)
            if (!Convert.IsDBNull(dv[i][1]))
            { kol++; sum += (int)dv[i][1]; }

        if (kol>0) str = string.Format("Среднее арифметическое = {0} {1}",
                         1.0*sum /kol,kol);
        else str = "Данные по полю kod не заполнены";

        MessageBox.Show(str,"Вариант 1");

        // ------------------------------------------
        // Вычисление среднего арифметического - Вариант 2
        // ------------------------------------------

        sum = 0;
        kol = 0;

        foreach (DataRowView row in dv)
            if (!Convert.IsDBNull(row[1]))
                { kol++; sum += (int)row[1]; }

        if (kol > 0) str = string.Format("Среднее арифметическое = {0} {1}",
                                         1.0 * sum / kol, kol);
        else str = "Данные по полю kod не заполнены";

        MessageBox.Show(str,"Вариант 2");

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

// Поиск записей
private void Find_Click(object sender, EventArgs e)
{
```

```
    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str;

        //Получение объекта DataView для поиска и сортировки
        DataView datav = ds.Tables[0].DefaultView;
        //datav.Sort = "kod";//устанавливается сортировка по выбранному полю
        //datav.Sort = "kod desc";
        //datav.Sort = "kod,Name";


        //int pos = bindingSource1.Find("Kod", 10);//выполняется медленно, не
        //требует сортировки
        //textBox1.Text - только число поля код
     // pos = -1 - значение не найдено

     //Find принимает массив значений в зависимости от сортировки (свойство Sort)
     //Разбор строки

        string[] mas_str = textBox1.Text.Split(new char[] {','},
                           StringSplitOptions.RemoveEmptyEntries);

        int pos = datav.Find(mas_str);
        if (pos >= 0)
            dataGridView1.CurrentCell = dataGridView1.Rows[pos].Cells[0];
        else
            MessageBox.Show("Значение не найдено");

        ////////////////////////////////////////
        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
                  1000.0).ToString() + " секунд";
        MessageBox.Show(str);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Primarykey - создание индекса (уникальное поле таблицы)
//Работа с фильтром!!!
//Фильтрация данных с помощью метода Select в таблице
//Фильтр работает с индексными файлами и без них.
//Фильтр с учетом индекса делается мгновенно!!!
private void Filter_Click(object sender, EventArgs e)
{

    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str = "";

       //Создание индексного файла по 1-му полю таблицы
       //При создании следующих индексов-все предыдущие сохраняются.
```

```
        //При создании индекса PrimaryKey значения в столбце должны быть уникальны

         //ds.Tables[0].PrimaryKey = new DataColumn[] { ds.Tables[0].Columns[0] };

         DataRow[] datar = ds.Tables[0].Select(textBox1.Text);
         DataTable table = ds.Tables[0];

         DataTable newTable = new DataTable();
         DataRow vr;
         int i;

         for (i = 0; i < table.Columns.Count; i++)
            newTable.Columns.Add(table.Columns[i].ColumnName,
                                 table.Columns[i].DataType);

         foreach (DataRow row in datar)
         {
             vr = newTable.NewRow();

             for (i = 0; i < table.Columns.Count; i++)
                 vr[i] = row[i];

             newTable.Rows.Add(vr);
         }


         bindingSource1.DataSource = newTable;
         dataGridView1.DataSource = bindingSource1;

         //Скрыть ненужные кнопки
         show(true, 2);

         //Запретить удаление, добавление и редактирование записей
         dataGridView1.AllowUserToAddRows = false;
         dataGridView1.AllowUserToDeleteRows = false;
         dataGridView1.EditMode = DataGridViewEditMode.EditProgrammatically;

         dt2 = DateTime.Now;
         tt = dt2 - dt1;

         str = str + "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds
                       / 1000.0).ToString() + " секунд";
         MessageBox.Show(str);
     }

     catch (Exception ex)
     {
         MessageBox.Show(ex.Message);
     }

}

//Отключение фильтра
private void Filter_off_Click(object sender, EventArgs e)
{
     bindingSource1.DataSource = ds.Tables[0];
     dataGridView1.DataSource = bindingSource1;

     //Показать кнопки и элеменнты в зависимости от выбранной таблицы
     if (baza.Substring(0, 3).ToUpper() == "OTL" && table.ToUpper() == "OTL_TAB")
          show();
     else show(true, 1);

     //Вернуть удаление, добавление и редактирование записей
     dataGridView1.AllowUserToAddRows = true;
```

```
    dataGridView1.AllowUserToDeleteRows = true;
    dataGridView1.EditMode = DataGridViewEditMode.EditOnKeystrokeOrF2;
}

//Возврат
private void Vozvrat_Click(object sender, EventArgs e)
{
    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str;
        //Снятие фильтра и сортировки
        ds.Tables[0].DefaultView.Sort = "";
        ds.Tables[0].DefaultView.RowFilter = "";
        Check_Sort.Checked = false;

        //Откат изменений
        ds.RejectChanges();

        ////////////////////////////////////////
        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
                1000.0).ToString() + " секунд";
        MessageBox.Show(str);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Возврат позиции текущей строки
private void ButtonRecno_Click(object sender, EventArgs e)
{
    try
    {
        MessageBox.Show("Строка № " + bindingSource1.Position);
        //MessageBox.Show("Строка № " + dataGridView1.CurrentRow.Index);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

// Сортировка записей по заданному условию
private void Check_Sort_CheckedChanged(object sender, EventArgs e)
{

    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str;
        if (Check_Sort.Checked == true)
            ds.Tables[0].DefaultView.Sort = textBox1.Text;
```

```
        else
            ds.Tables[0].DefaultView.Sort = "";

        ////////////////////////////////////////
        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
               1000.0).ToString() + " секунд";
        MessageBox.Show(str);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Функция создания индекса по выражению
private void CreateIndex_Click(object sender, EventArgs e)
{
    //Разбор строки для создания индексных файлов
    try
    {
        if (kol_index > 4)
        {
            MessageBox.Show("Количество индексов не должно превышать 5!");
            return;
        }

        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str;
        string[] mas_str = textBox1.Text.Split(new char[] {','},
                 StringSplitOptions.RemoveEmptyEntries);
        if (mas_str.Length<1) return;
        //Формирование массива DataColumn
        DataColumn[] datacol = new  DataColumn[mas_str.Length];
        for (int i = 0; i < mas_str.Length; i++)
        {
            datacol[i] = ds.Tables[0].Columns[mas_str[i].Trim()];
            if (datacol[i] == null)
            {
                MessageBox.Show("Ошибка в выражении");
                return;
            }
        }
        ////////////////////////////////////////////////////////////////

        ds.Tables[0].PrimaryKey = datacol;
        kol_index++;
        if (kol_index == 1) label4.Text = "Индексы: ";
        label4.Text += textBox1.Text.Trim()+"\n";
        ////////////////////////////////////////
        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
               1000.0).ToString() + " секунд";
        MessageBox.Show(str);
    }

    catch (Exception ex)
```

```
    {
        MessageBox.Show(ex.Message);
    }
}

// Формирование таблицы базы данных с заданными порциями записей
private void Formir_BD_Click(object sender, EventArgs e)
{
    int per_dob = 0;

    try
    {
        string str = textBox1.Text;
        per_dob = int.Parse(str);

        if (per_dob < 100 || per_dob > 1000000 ||
            (per_dob / 100.0 - per_dob / 100) > 0.0001)
        {
         if (MessageBox.Show
         ("ВВедите в к-во записей (кратных 100) для одной дополняемой порции",
        "Ошибка, Повторите ввод", MessageBoxButtons.OKCancel) == DialogResult.OK)
            {
                textBox1.Focus();
                label1.Text = "ВВедите число кратное 100";
            }
            else label1.Text = "";
            return;
        }
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }


        //Удаление записей
        if (cmd != null) cmd.Dispose();

        try
        {
            //Работа с транзакциями через объект OleDbCommand
            cmd = new OleDbCommand("DELETE FROM otl_tab", conn);
            cmd.Transaction = conn.BeginTransaction();
            //Выполнить без получения запроса
            cmd.ExecuteNonQuery();
            cmd.Transaction.Commit();
        }

        catch (Exception ex)
        {
            //Откат транзакции
            cmd.Transaction.Rollback();
            MessageBox.Show(ex.Message);
            return;
        }

        try
        {
            //Перечитать набор
            ds.Tables.Clear();
            da.Fill(ds);
            dataGridView1.DataSource = null;
            //dataGridView1.DataSource = ds.Tables[0];
```

```
            ////////////////////////////////////////

            // Добавление записей заданными порциями
           int povt = -1;
           string mes="";
           int kol_zapis = 0;
           int j = 0;

            while(true)
            {
            povt++;
            if(kol_zapis > 999000)break;
            if(povt!=0)
            {
    mes = string.Format("Будете добавлять еще {0} записей\nВ базе - {1} записей",
    per_dob,ds.Tables[0].Rows.Count);

            if (MessageBox.Show(mes,
               "Внимание!!!",MessageBoxButtons.YesNo) == DialogResult.No)break;
            }


            for(j = 1+povt*per_dob;j<=per_dob*(1+povt);j++)
                   {
                   kol_zapis++;

                //Формирование новой строки
                DataRow ins = ds.Tables[0].NewRow();

                ins[0] = ds.Tables[0].Rows.Count;
                ins[1] = j;
                ins[2] = "aaaa";
                ins[3] = DateTime.Now.Date;
                ins[4] = false;

                //Добавление новой строки в таблицу DataSet
                ds.Tables[0].Rows.Add(ins);

                }
            }//end while


            //Обновление записей в БД
            da.Update(ds.Tables[0]);
            label1.Text = "";

        }

        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }

        //Привязка таблицы к DataGridView

        bindingSource1.DataSource = ds.Tables[0];
        dataGridView1.DataSource = bindingSource1;

        kol_index = 0;
        label4.Text = "Индексы : нет";
    }

//Запись файла XML
private void ButtonSaveXML_Click(object sender, EventArgs e)
{
```

```
            saveFileDialog1.FileName = "1.xml";
            saveFileDialog1.Filter = "*.xml|*.xml";
            if (saveFileDialog1.ShowDialog() != DialogResult.OK) return;

            //Формирование объекта Connection
            try
            {
                ds.WriteXml(saveFileDialog1.FileName, XmlWriteMode.WriteSchema);
            }

            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        //Открытие и чтение файла XML
        private void ButtonLoadXML_Click(object sender, EventArgs e)
        {
            openFileDialog1.FileName = "1.xml";
            openFileDialog1.Filter = "*.xml|*.xml";
            if (openFileDialog1.ShowDialog() != DialogResult.OK) return;

            //Формирование объекта Connection
            try
            {

                ds.Tables.Clear();

                ds.ReadXml(openFileDialog1.FileName);
                dataGridView1.DataSource = ds.Tables[0];
            }

            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

Для отримання структури довільної бази даних використовується метод ***GetSchema()*** об'єкта **Connection**. Цей метод повертає об'єкт **DataTable**. Головним його параметром є параметр типу запиту схеми, який говорить про те, яку інформацію про структуру БД потрібно отримати. Так, якщо необхідно отримати інформацію про таблиці як параметр передамо рядок «Tables».

При виборі певної бази даних **MS ACCESS** здійснюється формування списку таблиць бази даних, реалізованого у функції ***Structura_BD()***.

Об'єкт **DataTable**, у якому зберігатиметься інформація про таблиці БД, оголошені як **table**.

Щоб відібрати потрібні таблиці з даними певної БД, необхідно перевірити тип таблиці. Якщо тип таблиці відповідає "TABLE" - цю таблицю необхідно додати до списку **listBox1**. У нашому випадку під час роботи з форматом **mdb**, об'єкт **DataTable** отримає дані про імена та типи таблиць. У таблиці 7.20 наведено інформацію про базу даних **otl.mdb**, яка повертається в об'єкт **DataTable** table.

З таблиці 7.20 видно, що у БД **otl.mdb** є 6 системних таблиць, причому 2 мають тип **ACCESS TABLE**, 4 - **SYSTEM TABLE**, одна таблиця з даними **otl_tab** (**TABLE**) та один SQL-запит - Запрос2 (тип **VIEW**).

Таким чином, для нашого випадку у листбоксі буде виведено лише одну таблицю **otl_tab**.

Таблиця 7.20

Структура бази даних otl.mdb

Им'я таблиці (TABLE_NAME)	**Тип таблиці (TABLE_TYPE)**
MSysAccessObjects	ACCESS TABLE
MSysAccessXML	ACCESS TABLE
MSysACEs	SYSTEM TABLE
MSysObjects	SYSTEM TABLE
MSysQueries	SYSTEM TABLE
MSysRelationships	SYSTEM TABLE
otl_tab	TABLE
Запрос2	VIEW

Після формування списку таблиць, у листбоксі ***listBox2*** виводиться список полів обраної таблиці. З цією метою додано відгук на натискання елемента зі списку вибору таблиць (об'єкт ***listBox1***), який називається ***listBox1_SelectedIndexChanged***.

Наведемо функцію відгуку щодо вибору відповідної таблиці зі списку:

```
// Вывод полей в листбокс по заданной таблице
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{

    string str;
    try
    {
        table = listBox1.SelectedItem.ToString();
        str = "Select * From [" + table + "]";

        if (da != null) da.Dispose();

        da = new OleDbDataAdapter(str, conn);
        ds.Tables.Clear();

        da.Fill(ds);

        listBox2.Items.Clear();
        DataTable table1 = ds.Tables[0];

        for (int i = 0; i < table1.Columns.Count; i++)
            listBox2.Items.Add(table1.Columns[i].ColumnName + "  -  " +
                table1.Columns[i].DataType);

    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }
```

```
        textBox1.Text = str;
        Zapros_Select_Click(null, EventArgs.Empty);
        groupBox1.Text = "Работа с таблицей " + table;
    }
```

У цій функції до лістбоксу ***listBox2*** виводиться інформація про поля обраної таблиці з зазначенням їх типів. Для цього доцільно повернути об'єкт **DataTable** вибраної таблиці та з використанням властивостей ***ColumnName*** та ***DataType()*** об'єкта ***Column*** повернути дані про ім'я та тип відповідного поля. Крім того, при виборі конкретної таблиці, до елементу керування ***DatagridView*** повертається результат SQL-запиту (функція ***Zapros_Select_Click()***).

У таблиці 7.21 наведено функціональне призначення користувальницьких функцій відгуків, які відповідають певним елементам керування.

Таблиця 7.21

Користувальницькі функції відгуків

Назва кнопки	Назва об'єкта	Назва функції	Призначення
Відкриття бази даних для MS ACCESS (*.mdb)			
Вибір бази	Vibor_BD	Vibor_BD_Click	Вибір бази даних для MS ACCESS (*.mdb)
Робота з SQL-запитами			
Запит на вибірку	Zapros_Select	Zapros_Select_Click	Формування об'єкта DataTable як результата запиту на вибірку
Запит на виконання	Zapros_Make	Zapros_Make_Click	Виконання змін у БД
Видалення, сортування, пошук, фільтрація, оновлення набору запитів			
Створення індексу	CreateIndex	CreateIndex _Click	Створення індексу
Видалення	Zapis_Delete	Zapis_Delete_Click	Видалення поточного запису
Оновлення записів БД	Update_BD	Update_BD_Click	Оновлення записів у базі даних
Сортування	Check_Sort	Check_Sort_ CheckedChanged	Сортування записів у наборі
Пошук	Find	Find_Click	Пошук значення у таблиці
Фільтр	Filter	Filter_Click	Фільтрування записів у наборі
Повернення	Vozvrat	Vozvrat_Click	Оновлення набору записів

Продовження таблиці 7.21

Назва кнопки	Назва об'єкта	Назва функції	Призначення
Робота з XML			
Зберегти у XML	ButtonSaveXML	ButtonSaveXML_Click	Збереження таблиці до файлу *.xml
Вибір XML	ButtonLoadXML	ButtonLoadXML_Click	Завантаження таблиці з файлу *.xml
Переміщення по записам			
First	ButtonFirst	ButtonFirst_Click	Перехід на перший запис
Last	ButtonLast	ButtonLast_Click	Перехід на останній запис
<----	ButtonLeft	ButtonLeft_Click	Перехід на попередній запис
---->	ButtonRight	ButtonRight_Click	Перехід на наступний запис
PgUp	ButtonPgup	ButtonPgup_Click	Перехід на попередню сторінку
PgDn	ButtonPgdn	ButtonPgdn_Click	Перехід на наступну сторінку
Recno	ButtonRecno	ButtonRecno_Click	Повернення абсолютного номера запису
Кнопки, призначені лише для роботи з otl_tab			
Додавання	Zapis_Add	Zapis_Add_Click	Додавання записів до таблиці otl_tab
Оновлення	Zapis_Update	Zapis_Update_Click	Оновлення поточного запису
Значення полів	Values_Fields	Values_Fields_Click	Виведення значень полів та розрахункові операції
Формування БД	Formir_BD	Formir_BD_Click	Формування таблиці бази даних із заданими порціями записів

Ці функції відгуків розбиті на незалежні групи, які відповідатимуть назвам розділів методичних вказівок. Слід зазначити, що останній розділ таблиці, "**Кнопки, призначені тільки для роботи з otl_tab**", включає роботу з функціями, які демонструють найбільш повну роботу з набором записів, наприклад додавання та оновлення записів. Розглянемо кожен розділ окремо.

Опишемо функцію вибору бази даних для MS ACCESS.

У цій функції вибираємо базу даних формату ***.mdb** і за допомогою методу **Open()** об'єкта **Connection** відкриваємо з'єднання. З'єднання відкривається за допомогою драйвера **OLEDB** (клас **OledbConnection**).

При відкритті довільної БД на екрані дисплея буде показано обмежену кількість елементів керування (**show(true,1)**).

На наступному етапі викликається функція виведення структури БД **Structura_BD()**. У цій функції ми отримуємо список таблиць вибраної бази даних.

Робота з SQL-запитами

У цьому розділі розглядаються функції відгуку на роботу з SQL-запитами. Це відповідно ***<Запит на вибірку>*** – функція ***Zapros_Select_Click()*** та ***<Запит на виконання>*** (оновлення, видалення, додавання записів тощо) – ***Zapros_Make_Click()***.

Розглянемо функцію відгуку ***Zapros_Select_Click()***, текст якої наведено нижче.

```
//Запрос на выборку
 private void Zapros_Select_Click(object sender, EventArgs e)
 {
     string str = textBox1.Text.ToUpper();

     if (str.Length<6||str.Substring(0, 6) != "SELECT")
     {
         MessageBox.Show("В запросе на выборку отсутствует параметр SELECT",
               "Ошибка!!!", MessageBoxButtons.OK, MessageBoxIcon.Information);
         textBox1.Text = "SELECT <*>|<список полей> FROM <список таблиц>";
         textBox1.Focus();
         return;

     }

     try
     {
         //Создание объекта Recordset
         if (da != null) da.Dispose();
         da = new OleDbDataAdapter(textBox1.Text, conn);
         ds.Tables.Clear();

         //Сохранение данных через объект OleDbCommandBuilder
         // Предусмотрено для удаления, обновления и добавления записей в БД
         // Наличие ключевого поля обязательно ! Учитывается уникальность записей.

         OleDbCommandBuilder bulder = new OleDbCommandBuilder(da);
         //Символы в которые будут заключены поля и таблицы БД
         //при формировании запросов
         bulder.QuotePrefix = "[";
         bulder.QuoteSuffix = "]";

         da.Fill(ds);

         //Привязка bindingSource к объекту DataTable
         //и dataGridView к объекту bindingSource

        bindingSource1.DataSource = ds.Tables[0];
        dataGridView1.DataSource = bindingSource1;

     //Показать элементы управления в зависимости от выбранной таблицы
    if (baza.Substring(0,3).ToUpper()=="OTL"&&table.ToUpper() == "OTL_TAB")show();
    else show(true, 1);
```

```
            label4.Text = "Индексы : нет";
            kol_index = 0;
            //Разрешить удаление, добавление и редактирование записей
            dataGridView1.AllowUserToAddRows = true;
            dataGridView1.AllowUserToDeleteRows = true;
            dataGridView1.EditMode = DataGridViewEditMode.EditOnKeystrokeOrF2;
        }

        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
```

У функції відгуку ***Zapros_Select_Click()*** формується запит на вибірку, результат якого записується в контейнер ***DataSet*** (об'єкт ***DataTable***) і якщо він сформований – виводиться в елемент керування ***DataGridView***.

Спочатку перевіряється наявність першого слова ***SELECT*** у текстовому вікні. Якщо ключове слово ***SELECT*** відсутнє, на екрані буде відображено відповідне повідомлення. Крім того, у текстове поле ***textBox1*** буде дано повний синтаксис команди ***SELECT***.

Формує набір записів метод ***Fill()*** об'єкта ***DataAdapter***. В якості параметрів об'єкта ***DataAdapter*** передається з'єднання ***Connection*** та текст SQL-запиту. Перед формуванням ***DataTable*** необхідно очистити контейнер таблиць ***DataSet*** за допомогою наступного рядка:

```
ds.Tables.Clear();
```

Об'єкт **CommandBuilder**, який прив'язується до **DataAdapter**, призначений для формування трьох запитів при оновленні бази даних на сервері. Це запити на видалення, оновлення та додавання даних до таблиці (DELETE, UPDATE, INSERT INTO).

Далі за допомогою об'єкта-зв'язувача ***bindingSource1*** прив'яжемо вибрану таблицю до об'єкта ***dataGridView1***.

```
            bindingSource1.DataSource = ds.Tables[0];
            dataGridView1.DataSource = bindingSource1;
```

Для відв'язування набору даних **DataTable** необхідно у властивість ***DataSource*** об'єкта ***dataGridView1*** передати «null».

```
            dataGridView1.DataSource = null;
```

При спрацьовуванні кожного запиту на вибірку здійснюється контроль обраної бази, таблиці та кількості полів обраної таблиці. Так якщо перші 3 символи бази даних відповідають "OTL" (otl.mdb для MS ACCESS), ім'я таблиці відповідає "OTL_TAB" - на екрані дисплея будуть показані всі елементи керування **show()**. Інакше матиме місце обмежена робота з таблицею обраної БД – ***show(true, 1)***. Другий параметр функції ***show()***, говорить про те, чи потрібно приховувати певні елементи керування. За замовчуванням цей параметр дорівнює 0, тобто буде показано всі елементи

керування. Якщо цей параметр відрізняється від нуля – на екран не виведуться такі кнопки: ***(<Додавання>***, ***<Оновлення>***, ***<Значення полів>***, ***<Формування БД>***).

Слід зазначити, що в даному випадку, як і в попередніх варіантах, використовується технологія ***try ... catch()*** для обробки помилок.

Розглянемо функцію відгуку ***Zapros_Make_Click()*** (кнопка ***<Запит на виконання>***), в якій виконуються SQL-запити, пов'язані зі зміною даних у таблиці (SQL-запити на виконання, за винятком SQL-запитів на вибірку). Текст функції ***Zapros_Make_Click()*** наведено нижче.

```
        // Запрос на выполнение
        private void Zapros_Make_Click(object sender, EventArgs e)
        {

            //Проверка времени выполнения запроса
            DateTime dt1, dt2;
            TimeSpan tt;
            dt1 = DateTime.Now;
            ////////////////////////////////////////
            string str = textBox1.Text.ToUpper().Trim();
            string str1="";
            int kod = 0;
            label1.Text = "";

          if (str.Length > 2 && str.Substring(0, 3) == "SEL")
              { MessageBox.Show("Команда SELECT недопустима для данной кнопки"); return;}

            if (str.Length > 2 && str.Substring(0, 3) == "DEL")
            {
                str1 = "DELETE FROM <таблица> [ where<условие> ]";
                kod = 1;
            }
            if (str.Length > 2 && str.Substring(0, 3) == "UPD")
            {
                str1 = "UPDATE <таблица> SET <поле>=<выражение>, [< поле>=<выражение>]...
[where<условие>]";
                kod = 2;
            }

            if (str.Length > 2 && str.Substring(0, 3) == "INS")
            {
                str1 = "INSERT INTO <таблица> (<список имен полей>) VALUES (<список
значений>)";
                kod = 3;
            }

            if (str.Length > 2 && str.Substring(0, 3) == "CRE")
            {
                str1 = "CREATE TABLE <таблица> (<имя поля1> <тип>, <имя поля2> <тип>...
)";
                kod = 4;
            }

            if (str.Length > 2 && str.Substring(0, 3) == "DRO")
            {
                str1 = "DROP TABLE <таблица>";
                kod = 5;
            }

            if (kod>0) label1.Text = "Синтаксис: " + str1;
```

```
        try
        {
            if (cmd != null) cmd.Dispose();
            //Работа с транзакциями через объект OleDbCommand
            cmd = new OleDbCommand(textBox1.Text, conn);
            cmd.Transaction = conn.BeginTransaction();
               //Выполнить без получения запроса
               cmd.ExecuteNonQuery();
            cmd.Transaction.Commit();
        }

        catch (Exception ex)
        {
            //Откат транзакции
            cmd.Transaction.Rollback();
            MessageBox.Show(ex.Message);

            if (kod > 0) textBox1.Text = str1;
            textBox1.Focus();
            return;
        }

        //Изменить запрос (перечитать набор)
        ds.Tables.Clear();

        da.Fill(ds);

        bindingSource1.DataSource = ds.Tables[0];
        dataGridView1.DataSource = bindingSource1;

        label4.Text = "Индексы: нет";
        kol_index = 0;

        ////////////////////////////////////////
        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
                 1000.0).ToString() + " секунд";
        MessageBox.Show(str);

        textBox1.Text = "";
        label1.Text = "";
    }
```

При реалізації функції ***Zapros_Make_Click()*** здійснюється виконання SQL-запитів на зміну даних. Якщо користувач почав використовувати команду SELECT (перевіряється лише перші 3 літери), буде виведено повідомлення про неприпустимість використання цієї кнопки. У цій функції передбачено виведення синтаксису для певних команд SQL у текстове поле. У разі, якщо допущено помилку у тексті SQL-запиту, у ***textBox1*** буде виведено синтаксис команди. Синтаксис наведено на наступні команді SQL: DELETE (запит на видалення), UPDATE (запит на оновлення), INSERT INTO (запит на додавання), CREATE TABLE (запит на створення таблиці), DROP TABLE (запит на видалення таблиці).

У цьому фрагменті інтерес представляє робота з транзакцією, яку доцільно використовувати під час роботи з груповими запитами на зміну даних. Метод ***BeginTransaction()*** об’єкта ***Connection*** починає роботу з

транзакцією (властивість ***Transaction*** об'єкта ***Command***), а метод ***Commit()*** завершує та здійснює контроль над усіма операціями для даної транзакції. У разі помилки відбувається відкат транзакції за допомогою методу ***Rollback()***.

Після реалізації запиту на виконання необхідно перечитати набір, щоб у об'єкті ***DataGridView*** відобразилися змінені дані. Набір перечитується за допомогою повторного SQL-запиту на вибірку (метод ***Fill()*** об'єкта ***DataAdapter***).

Увага!!! Особливістю цієї та багатьох інших функцій є контроль її виконання за певний час, з точністю до мілісекунд. Даний підхід реалізований за допомогою класичних структур DateTime та TimeSpan. Експерименти виконання кожної функції за певний час дають змогу проаналізувати роботу методів основних об'єктів ADO.NET та ефективно побудувати свою роботу.

У змінні ***dt1*** та ***dt2*** структури ***DateTime*** записується поточна дата та час, відповідно на початку виконання функції та наприкінці. Різниця цих змінних і дасть нам кількість секунд на експериментальні операції з урахуванням мілісекунду (структура ***TimeSpan***).

Видалення, сортування, пошук, фільтрація, оновлення набору записів

У цьому розділі розглядаються функції відгуків створення індексу (***CreateIndex_Click()***, кнопка ***<Створення індексу>***), видалення (***Zapis_Delete_Click()***, кнопка ***<Видалення>***), оновлення записів БД (***Update_BD_Click()***, кнопка ***<Оновлення записів БД>***), сортування (***Che ()***, прапорець ***<Сортування>***), пошуку (***Find_Click()***, кнопка ***<Пошук>***), фільтрації (***Filter_Click()***, кнопка ***<Фільтр>***) оновлення (***Vozvrat_Click()***, кнопка ***<Повернення>***) набору записів.

Розглянемо функцію відгуку ***CreateIndex_Click()***, за допомогою якої створюються індекси. Текст цієї функції наведено нижче.

```
//Функция создания индекса по выражению
private void CreateIndex_Click(object sender, EventArgs e)
{
    //Разбор строки для создания индексных файлов
    try
    {
        if (kol_index > 4)
        {
            MessageBox.Show("Количество индексов не должно превышать 5!");
            return;
        }

        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str;
        string[] mas_str = textBox1.Text.Split(new char[] {','},
                 StringSplitOptions.RemoveEmptyEntries);
        if (mas_str.Length<1) return;
        //Формирование массива DataColumn
        DataColumn[] datacol = new  DataColumn[mas_str.Length];
        for (int i = 0; i < mas_str.Length; i++)
```

```
            {
                datacol[i] = ds.Tables[0].Columns[mas_str[i].Trim()];
                if (datacol[i] == null)
                {
                    MessageBox.Show("Ошибка в выражении");
                    return;
                }
            }
            ////////////////////////////////////////////////////////////

            ds.Tables[0].PrimaryKey = datacol;

            kol_index++;
            if (kol_index == 1) label4.Text = "Индексы: ";
            label4.Text += textBox1.Text.Trim()+"\n";
            ////////////////////////////////////////
            dt2 = DateTime.Now;
            tt = dt2 - dt1;

            str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
                    1000.0).ToString() + " секунд";
            MessageBox.Show(str);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
```

Ці індекси призначені для прискорення роботи на стороні клієнта з локальним набором даних **DataTable**. Властивість для створення індексу ***PrimaryKey***. Входом є масив об'єктів **DataColumn[]**.

```
            ds.Tables[0].PrimaryKey = datacol;
```

Спочатку за допомогою методу **Split()** класу **string** одержуємо масив імен полів ***mas_str***. Потім за допомогою циклу створюємо масив індексів **DataColumn []** з ім'ям ***datacol***.

Ці індекси впливають на швидкий пошук, сортування та фільтрацію даних з використанням об'єкта ***DataTable***.

Розглянемо функцію відгуку ***Zapis_Delete_Click()***, за допомогою якої видаляється поточний запис набору даних. Текст цієї функції наведено нижче.

```
// Удаление текущей записи
private void Zapis_Delete_Click(object sender, EventArgs e)
{
    try
    {

        DataRowView dr = (DataRowView) bindingSource1.Current;
        dr.Delete();

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }

}
```

Властивість ***Current*** об'єкта-зв'язувача ***bindingSource1*** повертає клас рядка ***DataRowView***, де є метод ***Delete()*** для видалення поточного запису. Слід зазначити, що з об'єкта ***DataTable*** цей запис не видаляється, а лише позначається на видалення. При видаленні запису відбувається перебудова об'єкта представлення ***DataView***, який входить до об'єкта ***DataTable***.

Далі розглянемо функцію відгуку на оновлення записів у БД ***Update_BD_Click()***, за допомогою якої відбувається оновлення даних у БД на стороні сервера. Наведемо текст цієї функції.

```
 //Обновление записей базы данных (до этого все изменения происходят только в
 //DataGridView)
private void Update_BD_Click(object sender, EventArgs e)
{
    try
    {
        da.Update(ds.Tables[0]);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Метод ***Update()*** об'єкта ***DataAdapter*** для кожного зміненого, доданого чи позначеного на видалення запису посилає відповідні запити до таблиці БД на бік сервера. Це запити видалення (команда DELETE), оновлення (команда UPDATE), додавання даних (команда INSERT INTO). Дані три запити формує об'єкт ***CommandBuilder*** за ключовим полем таблиці бази даних.

Увага! За відсутності ключового поля в таблиці БД – технологія ADO.NET працювати не буде, тому що не буде зрозуміло, як формувати запити на оновлення даних щодо кожного запису. Ці запити формуються виключно ключовим полем.

Розглянемо функцію відгуку на прапорець ***Check_Sort***, ***Check_Sort_CheckedChanged()***, у якій здійснюється сортування за заданими полями вибраної таблиці. Наведемо текст цієї функції.

```
// Сортировка записей по заданному условию
private void Check_Sort_CheckedChanged(object sender, EventArgs e)
{

    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str;
        if (Check_Sort.Checked == true)
            ds.Tables[0].DefaultView.Sort = textBox1.Text;
        else
            ds.Tables[0].DefaultView.Sort = "";

        ////////////////////////////////////////
        dt2 = DateTime.Now;
```

```
            tt = dt2 - dt1;

            str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
                        1000.0).ToString() + " секунд";
            MessageBox.Show(str);
        }

        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
```

Сортування не впливає на об'єкт **DataTable**, лише на подання даних таблиці. У цій функції використовується властивість ***Sort*** об'єкта **DataView**, отриманого за допомогою властивості **DefaultView**. Для призначення сортування необхідно виконати наступний рядок.

```
ds.Tables[0].DefaultView.Sort = textBox1.Text;
```

Головний сенс властивості ***Sort*** об'єкта **DataView** полягає у його використанні для швидкого пошуку відповідних даних із використанням методу ***Find()***.

Розглянемо функцію відгуку ***Find_Click()*** (кнопка ***<Пошук>***), де детально описується призначення і функціональні особливості методу ***Find()***. Текст цієї функції наведено нижче.

```
// Поиск записей
private void Find_Click(object sender, EventArgs e)
{
    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str;

        //Получение объекта DataView для поиска и сортировки
        DataView datav = ds.Tables[0].DefaultView;
        //datav.Sort = "kod";//устанавливается сортировка по выбранному полю
        //datav.Sort = "kod desc";
        //datav.Sort = "kod,Name";


        //int pos = bindingSource1.Find("Kod", 10);
        //выполняется медленно, не требует сортировки
        //textBox1.Text - только число поля код
        // pos = -1 - значение не найдено

     //Find принимает массив значений в зависимости от сортировки (свойство Sort)
     //Разбор строки

        string[] mas_str = textBox1.Text.Split(new char[] {','},
                 StringSplitOptions.RemoveEmptyEntries);

        int pos = datav.Find(mas_str);
        if (pos >= 0)
            dataGridView1.CurrentCell = dataGridView1.Rows[pos].Cells[0];
        else
```

```
            MessageBox.Show("Значение не найдено");

        ////////////////////////////////////////
        dt2 = DateTime.Now;
        tt = dt2 - dt1;

        str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
               1000.0).ToString() + " секунд";
        MessageBox.Show(str);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Метод ***Find()*** об'єкта **DataView** працює лише за наявності сортування та здійснює пошук на точний збіг даних. Перевагою пошуку є те, що його можна проводити відразу по кількох полях. Як вхідний параметр у функцію можна передати як об'єкт, так і масив об'єктів. У цьому фрагменті ми передаємо масив рядків ***mas_str[]***, отриманий за допомогою методу ***Split()*** класу **string**. Метод ***Find()*** повертає абсолютну позицію знайденого рядка, якщо рядок не знайдено число -1.

У деяких випадках роботи із набором записів доцільно відібрати дані за певною умовою. Для цього використовується фільтрація даних. Нижче наведемо функцію відгуку ***Filter_Click()*** (кнопка **<*Фільтр*>**).

```
//Primarykey - создание индекса (уникальное поле таблицы)
//Работа с фильтром!!!
//Фильтрация данных с помощью метода Select в таблице
//Фильтр работает с индексными файлами и без них.
//Фильтр с учетом индекса делается мгновенно!!!
private void Filter_Click(object sender, EventArgs e)
{

    try
    {
        DateTime dt1, dt2;
        TimeSpan tt;
        dt1 = DateTime.Now;
        ////////////////////////////////////////
        string str = "";

       //Создание индексного файла по 1-му полю таблицы
       //При создании следующих индексов-все предыдущие сохраняются.
       //При создании индекса PrimaryKey значения в столбце должны быть уникальны

        //ds.Tables[0].PrimaryKey = new DataColumn[] { ds.Tables[0].Columns[0] };

        DataRow[] datar = ds.Tables[0].Select(textBox1.Text);
        DataTable table = ds.Tables[0];

        DataTable newTable = new DataTable();
        DataRow vr;
        int i;

        for (i = 0; i < table.Columns.Count; i++)
           newTable.Columns.Add(table.Columns[i].ColumnName,
                                table.Columns[i].DataType);
```

```
            foreach (DataRow row in datar)
            {

                vr = newTable.NewRow();

                for (i = 0; i < table.Columns.Count; i++)
                    vr[i] = row[i];

                newTable.Rows.Add(vr);
            }

            bindingSource1.DataSource = newTable;
            dataGridView1.DataSource = bindingSource1;

            //Скрыть ненужные кнопки
            show(true, 2);

            //Запретить удаление, добавление и редактирование записей
            dataGridView1.AllowUserToAddRows = false;
            dataGridView1.AllowUserToDeleteRows = false;
            dataGridView1.EditMode = DataGridViewEditMode.EditProgrammatically;

            dt2 = DateTime.Now;
            tt = dt2 - dt1;

            str = str + "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds
                        / 1000.0).ToString() + " секунд";
            MessageBox.Show(str);
        }

        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }

    }
```

Слід зазначити, що існують різні способи сортування, пошуку та фільтрації даних. Усі вони з детальним аналізом наведено у прикладі **ADO_SortFilterFind**, розглянутому вище.

У цій функції наведено один зі способів фільтрації за допомогою методу ***Select()*** об'єкта **DataTable**. Цей спосіб фільтрації є найшвидшим та використовує індексні файли, створені за допомогою властивості ***PrimaryKey*** об'єкта **DataTable**.

Оскільки метод ***Select()*** повертає масив рядків **DataRow []**, то для їх відображення необхідно створити нову таблицю, та потім прив'язати її до **DataGridView**.

Щоб зняти фільтр, реалізований за допомогою методу **Select()**, необхідно просто прив'язати до елемента **dataGridView1** вихідну таблицю даних. Код функції зняття фільтра наведено нижче.

```
//Отключение фильтра
private void Filter_off_Click(object sender, EventArgs e)
{
    bindingSource1.DataSource = ds.Tables[0];
    dataGridView1.DataSource = bindingSource1;

    //Показать кнопки и элеменнты в зависимости от выбранной таблицы
    if (baza.Substring(0, 3).ToUpper() == "OTL" && table.ToUpper() == "OTL_TAB")
```

```
            show();
        else show(true, 1);

        //Вернуть удаление, добавление и редактирование записей
        dataGridView1.AllowUserToAddRows = true;
        dataGridView1.AllowUserToDeleteRows = true;
        dataGridView1.EditMode = DataGridViewEditMode.EditOnKeystrokeOrF2;
    }
```

Повернення до вихідної таблиці даних, отриманої в результаті SQL-запиту на вибірку, відбувається у функції відгуку ***Vozvrat_Click()*** (кнопка **<*Повернення*>**), текст якої наведено нижче.

```
    //Возврат
    private void Vozvrat_Click(object sender, EventArgs e)
    {
        try
        {
            DateTime dt1, dt2;
            TimeSpan tt;
            dt1 = DateTime.Now;
            ////////////////////////////////////////
            string str;
            //Снятие фильтра и сортировки
            ds.Tables[0].DefaultView.Sort = "";
            ds.Tables[0].DefaultView.RowFilter = "";
            Check_Sort.Checked = false;

            //Откат изменений
            ds.RejectChanges();

            ////////////////////////////////////////
            dt2 = DateTime.Now;
            tt = dt2 - dt1;

            str = "Запрос выполнен в течение " + (tt.Seconds + tt.Milliseconds /
                   1000.0).ToString() + " секунд";
            MessageBox.Show(str);
        }


        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
```

Для повернення початкового набору необхідно зняти сортування та фільтрацію даних об’єкта представлення **DataView**, а потім виконати відкат змін за допомогою методу ***RejectChanges()*** об’єкта **DataSet** (або **DataTable**). Такий підхід зручний у порівнянні з технологією **ADO**, тому що там потрібно було повторно виконати запит на вибірку, а тут ми повертаємося до початкового стану набору в об’єкті **DataTable**, тому що він зберігає початковий стан до виклику методу ***Update()*** об’єкта **DataAdapter**, або до застосування методу ***AcceptChanges()*** об’єктів **DataSet** або **DataTable**.

Робота з XML

У цьому розділі здійснюється запис даних до файлів *.xml, і навіть їх читання. Такий універсальний підхід у **ADO.NET** призначений імпорту та експорту даних між таблицями різних БД, і навіть передачі даних. Якщо у технології **ADO** файл ****.xml*** працював через провайдер і був аналогією БД, то це уніфікований формат для обміну даних.

Розглянемо функцію відгуку ***ButtonSaveXML_Click()*** (кнопка **<Зберегти у XML>**), в якій здійснюється запис об'єкта **DataSet** у файл ***.xml**. Текст цієї функції наведено нижче.

```
//Запись файла XML
private void ButtonSaveXML_Click(object sender, EventArgs e)
{
    saveFileDialog1.FileName = "1.xml";
    saveFileDialog1.Filter = "*.xml|*.xml";
    if (saveFileDialog1.ShowDialog() != DialogResult.OK) return;

    try
    {
        ds.WriteXml(saveFileDialog1.FileName, XmlWriteMode.WriteSchema);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Запис даних, включаючи їх структуру, відбувається за допомогою методу ***WriteXml()*** об'єкта **DataSet**. Параметр перерахування **XmlWriteMode WriteSchema** говорить про те, що крім даних у файл ****.xml*** буде записана структура таблиць.

Для читання даних розглянемо функцію відгуку ***ButtonLoadXML_Click()*** (кнопка ***<Вибір XML>***), текст якої наведено нижче.

```
//Открытие и чтение файла XML
private void ButtonLoadXML_Click(object sender, EventArgs e)
{
    openFileDialog1.FileName = "1.xml";
    openFileDialog1.Filter = "*.xml|*.xml";
    if (openFileDialog1.ShowDialog() != DialogResult.OK) return;

    try
    {

        ds.Tables.Clear();
        ds.ReadXml(openFileDialog1.FileName);
        dataGridView1.DataSource = ds.Tables[0];
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }

}
}
```

Тут за допомогою функції ***ReadXml()*** об'єкта **DataSet** здійснюється читання файлу ****.xml***, а потім його подальша прив'язка до **DataGridView1**.

Переходи по записах

У цьому розділі розглядаються функції відгуків з використанням класичних методів переходу по записах: ***ButtonFirst_Click()*** (кнопка ***<First>***) – перехід на перший запис; ***ButtonLast_Click()*** (кнопка ***<Last>***) – перехід на останній запис; ***ButtonLeft_Click()*** (кнопка ***<----***) – перехід на попередній запис; ***ButtonRight_Click() (***кнопка ***---->***) – перехід на наступний запис. Крім цього, розглядаються функції посторінкового переходу по записах: ***ButtonPgup_Click()*** (кнопка ***<PgUp>***) – перехід на попередню сторінку; ***ButtonPgdn_Click()*** (кнопка ***<PgDn>***) – перехід на наступну сторінку, а також функції встановлення абсолютного номера запису (номери по порядку, починаючи з нуля) – ***ButtonRecno_Click()*** (кнопка ***<Recno>***).

Наведемо функції відгуків переходу по записах, відповідно на перший, останній, попередній та наступний записи.

```
//Переход на первую запись
private void ButtonFirst_Click(object sender, EventArgs e)
{
    try
    {

        bindingSource1.MoveFirst();
        //dataGridView1.CurrentCell = dataGridView1.Rows[0].Cells[0];
        //bindingSource1.Position = 0; 2- й вариант

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Переход на предыдущую запись
private void ButtonLeft_Click(object sender, EventArgs e)
{
    try
    {
        if (bindingSource1.Position == 0) return;
        bindingSource1.MovePrevious();
        dataGridView1.CurrentCell =
              dataGridView1.Rows[bindingSource1.Position].Cells[0];
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

//Переход на следующую запись
private void ButtonRight_Click(object sender, EventArgs e)
{
    try
    {
        if (bindingSource1.Position == bindingSource1.Count - 1) return;
```

```
            bindingSource1.MoveNext();
            dataGridView1.CurrentCell =
                    dataGridView1.Rows[bindingSource1.Position].Cells[0];
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    //Переход на последнюю запись
    private void ButtonLast_Click(object sender, EventArgs e)
    {
        try
        {
            bindingSource1.MoveLast();
            dataGridView1.CurrentCell =
                        dataGridView1.Rows[bindingSource1.Position].Cells[0];
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
```

Переходи по записах здійснюються за допомогою об'єкта **bindingSource1**, у якому містяться такі функции: ***MoveFirst()*** – перехід до першого запису; ***MoveLast()*** – перехід на останній запис; ***MovePrevios()*** – перехід на попередній запис; ***MoveNext()*** – перехід на наступний запис.

Крім того, використовуються функції переходу на заданий логічний номер запису (абсолютне значення) та функції переходу по сторінках (групі записів). Тобто, знаючи кількість даних, що відображаються на екрані дисплея в елементі керування **DataGridView**, можна переміщатися на довільну сторінку з даними. Відповідні функції відповіді наведені нижче.

```
    //Возврат позиции текущей строки
    private void ButtonRecno_Click(object sender, EventArgs e)
    {
        try
        {
            MessageBox.Show("Строка № " + bindingSource1.Position);
            //MessageBox.Show("Строка № " + dataGridView1.CurrentRow.Index);
        }

        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    //Постраничный переход на 12 записей вверх
    private void ButtonPgup_Click(object sender, EventArgs e)
    {
        try
        {
            int idx = bindingSource1.Position - 12;
            idx = (idx<0)?0:idx;
            dataGridView1.CurrentCell = dataGridView1.Rows[idx].Cells[0];
        }
        catch (Exception ex)
        {
```

```
            MessageBox.Show(ex.Message);
        }
    }
    //Постраничный переход на 12 записей вниз
    private void ButtonPgdn_Click(object sender, EventArgs e)
    {
        try
        {
            int idx = bindingSource1.Position + 12;
            idx = (idx > bindingSource1.Count - 1) ? bindingSource1.Count - 1 : idx;
            dataGridView1.CurrentCell = dataGridView1.Rows[idx].Cells[0];
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
```

Перехід по абсолютним значенням записів у кожній з цих функцій здійснюється за допомогою властивості ***Position*** об'єкта **bindingSource1**.

Кнопки, призначені лише для роботи з otl_tab

Цей розділ призначений для роботи лише з даними експериментальної таблиці otl_tab, яка входить до БД otl.mdb. Тут розглядаються функції відгуків додавання (***Zapis_Add_Click()*** (кнопка ***<Додати>***), оновлення поточного запису (***Zapis_Update_Click()*** (кнопка ***<Оновлення>***), а також формування записів таблиці otl_tab з заданими порціями (***Formir_BD_Click()***) (кнопка ***<Формування БД>***)) та виведення розрахункових операцій по таблиці otl_tab (***Values_Fields_Click()*** (кнопка ***<Значення полів>***)).

Додавання записів здійснюється за допомогою методу ***Add()*** колекції рядків **DataRowCollection**. Наведемо текст коду функції ***Zapis_Add_Click()***.

```
    // Добавление записей в таблицу otl_tab
    private void Zapis_Add_Click(object sender, EventArgs e)
    {
        try
        {
            //Формирование новой строки
            DataRow ins = ds.Tables[0].NewRow();

            ins[0] = 12345;
            ins[1] = 56789;
            ins[2] = "Привет12345";
            ins[3] = DateTime.Now;
            ins[4] = true;

            //Добавление новой строки в таблицу DataSet
            ds.Tables[0].Rows.Add(ins);

            //Переход на нужную ячейку в dataGridView1

            dataGridView1.CurrentCell =
            dataGridView1.Rows[ds.Tables[0].Rows.IndexOf(ins)].Cells[0];

        }
        catch (Exception ex)
```

```
    {
        MessageBox.Show(ex.Message);
    }
}
```

Спочатку ми створюємо у пам'яті новий рядок **Row** зі структурою даних вибраної таблиці з допомогою методу ***NewRow()*** об'єкта **DataTable**. Потім заповнюємо її необхідними даними та за допомогою методу ***Add()*** додаємо її до таблиці.

Оновлення записів здійснюється у функції відгуку ***Zapis_Update_Click()***. Наведемо текст цієї функції.

```
// Обновление текущей записи
private void Zapis_Update_Click(object sender, EventArgs e)
{
    try
    {
        DataRow zap = ((DataRowView)(bindingSource1.Current)).Row;
        zap[0] = 54321;
        zap[1] = 98765;
        zap[2] = "Привет98765";
        zap[3] = DateTime.Now;
        zap[4] = false;

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Для зміни даних поточного запису необхідно отримати об'єкт – **DataRow**. Поточний запис повертає властивість **Current** об'єкта **bindingSource1**. Після отримання поточного запису дані просто потрібно змінити.

Працюючи з експериментальною таблицею otl_tab часто буває потрібно сформувати її заново. У цьому прикладі представлено функцію ***Formir_BD_Click()*** на кнопці ***<Формування БД>***, у якій відбувається формування таблиці з заданими порціями даних. Кожна порція даних має бути кратна 100 записів.

Текст цієї функції наведено нижче.

```
// Формирование таблицы базы данных с заданными порциями записей
private void Formir_BD_Click(object sender, EventArgs e)
{
    int per_dob = 0;

    try
    {
        string str = textBox1.Text;
        per_dob = int.Parse(str);

        if (per_dob < 100 || per_dob > 1000000 ||
            (per_dob / 100.0 - per_dob / 100) > 0.0001)
```

```
    {
        if (MessageBox.Show
    ("ВВедите в к-во записей (кратных 100) для одной дополняемой порции",
    "Ошибка, Повторите ввод", MessageBoxButtons.OKCancel) == DialogResult.OK)
        {
            textBox1.Focus();
            label1.Text = "ВВедите число кратное 100";
        }
        else label1.Text = "";
        return;
    }
}

catch (Exception ex)
{
    MessageBox.Show(ex.Message);
    return;
}


    //Удаление записей
    if (cmd != null) cmd.Dispose();

    try
    {
        //Работа с транзакциями через объект OleDbCommand
        cmd = new OleDbCommand("DELETE FROM otl_tab", conn);
        cmd.Transaction = conn.BeginTransaction();
        //Выполнить без получения запроса
        cmd.ExecuteNonQuery();
        cmd.Transaction.Commit();
    }

    catch (Exception ex)
    {
        //Откат транзакции
        cmd.Transaction.Rollback();
        MessageBox.Show(ex.Message);
        return;
    }

    try
    {
        //Перечитать набор
        ds.Tables.Clear();
        da.Fill(ds);
        dataGridView1.DataSource = null;
        //dataGridView1.DataSource = ds.Tables[0];
        ////////////////////////////////////////


        // Добавление записей заданными порциями
       int povt = -1;
       string mes="";
       int kol_zapis = 0;
       int j = 0;

        while(true)
        {
        povt++;
        if(kol_zapis > 999000)break;
```

```
            if(povt!=0)
            {
        mes = string.Format("Будете добавлять еще {0} записей\nВ базе - {1} записей",
                per_dob,ds.Tables[0].Rows.Count);

            if (MessageBox.Show(mes,
               "Внимание!!!",MessageBoxButtons.YesNo) == DialogResult.No)break;
            }


            for(j = 1+povt*per_dob;j<=per_dob*(1+povt);j++)
                   {
                   kol_zapis++;

                //Формирование новой строки
                DataRow ins = ds.Tables[0].NewRow();

                ins[0] = ds.Tables[0].Rows.Count;
                ins[1] = j;
                ins[2] = "aaaa";
                ins[3] = DateTime.Now.Date;
                ins[4] = false;

                //Добавление новой строки в таблицу DataSet
                ds.Tables[0].Rows.Add(ins);

                }
            }//end while


            //Обновление записей в БД
            da.Update(ds.Tables[0]);
            label1.Text = "";
        }

        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }

        //Привязка таблицы к DataGridView

        bindingSource1.DataSource = ds.Tables[0];
        dataGridView1.DataSource = bindingSource1;

        kol_index = 0;
        label4.Text = "Индексы : нет";
    }
```

Спочатку в змінну *per_dob* типу *int* записується значення рядка, введене у текстове поле ***textBox1***. В даному випадку для перетворення рядка на число використовується функція ***Parse()*** типу *int*.

У тому випадку, якщо кількість записів у порції менше 100 або більше 1000000, або введено число порції записів не кратне 100 – виведеться відповідне повідомлення про помилку і буде здійснено вихід з функції.

На наступному етапі видаляються всі записи таблиці *otl_tab*, використовуючи SQL-запит на видалення. Потім робиться відв'язування об'єкта ***DataTable*** від елемента керування ***DatagridView***.

Після цього здійснюється додавання записів заданими порціями. З цією метою запроваджено змінну **povt**, яка відповідає за кількість порцій даних. У вічному циклі ***while (true)*** вона постійно накопичується. До початку цикла їй дається значення -1. Потім вона накопичується і набуває значення 0. У циклі ***while()*** розташований "хитрий" цикл *for()* за допомогою якого додаються записи однієї порції. Він виглядає так:

```
for(j = 1+povt*per_dob;j<=per_dob*(1+povt);j++)
 {
   // тело цикла
 }
```

У цьому циклі змінна *per_dob* – це кількість записів заданої порції. Так, спочатку змінна *povt* дорівнює нулю, тобто буде записана перша порція даних (від 1 до *per_dob*), потім, коли записуватиметься друга порція даних - змінна *povt* дорівнює 1 (від 1 + *per_dob* до 2 * *per_dob*) тощо. При кожній новій порції даних, починаючи з другої, на екран виводитиметься повідомлення про виведення нової порції записів, де користувачеві надається можливість або додати чергову порцію записів, або відмовитися. При відмові буде здійснено вихід з вічного циклу ***while(true)***. Крім того, якщо кількість записів перевищить 999000, також буде здійснено вихід з вічного циклу. Всі записи до цієї функції додаються за допомогою методу ***Add ()***. Наприкінці виконання функції здійснюється оновлення записів БД, і навіть прив'язка об'єкта **DataTable** до елементу керування **DatagridView**.

За допомогою кнопки ***<Значення полів>*** (функція відгуку ***Values_Fields_Click()***) виводяться значення полів поточного запису, а також здійснюються арифметичні операції над полями таблиці *otl_tab*.

Текст цієї функції відгуку наведено нижче.

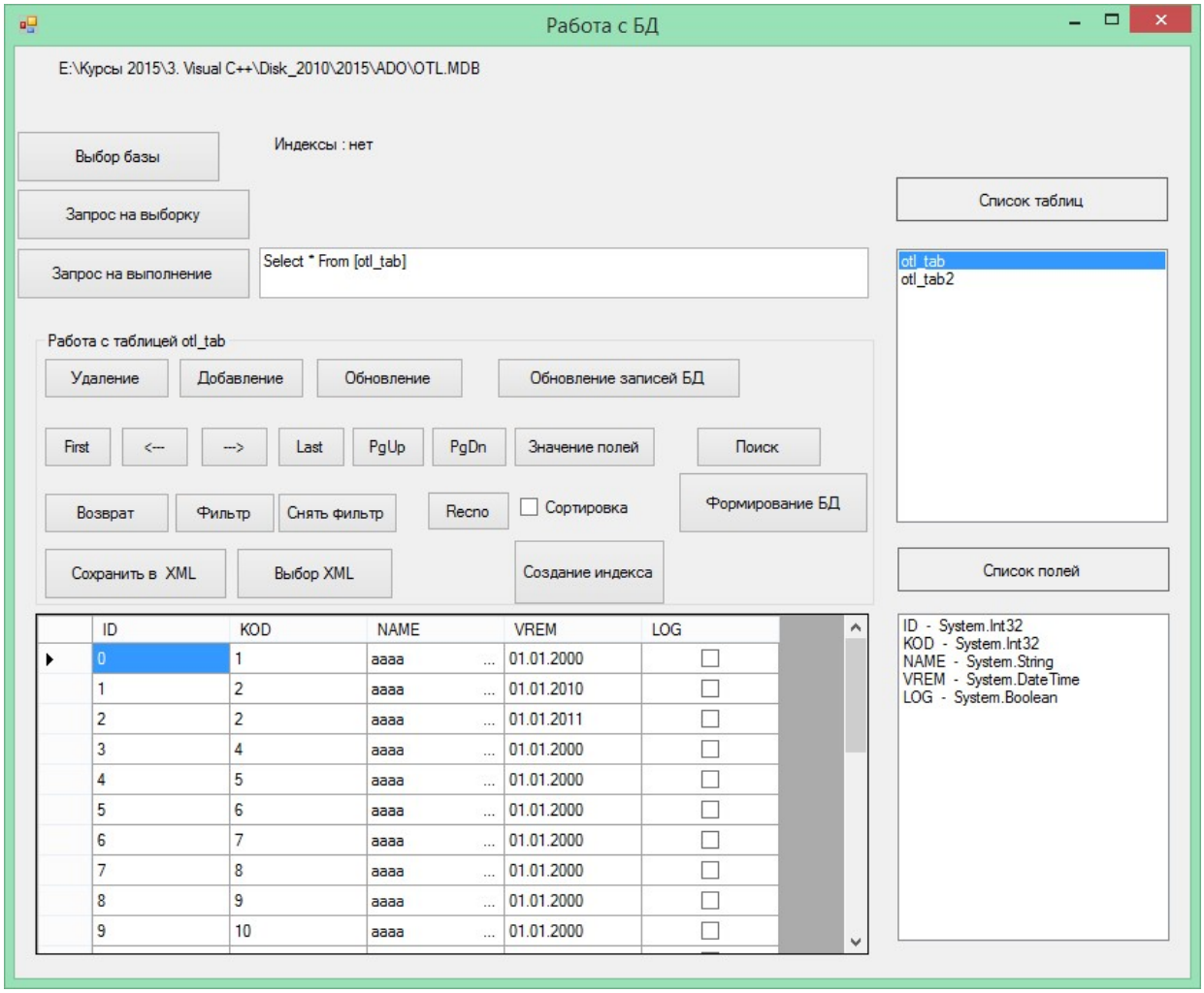
```
//Вывод значений полей и расчетные операции по представлению данных таблицы
private void Values_Fields_Click(object sender, EventArgs e)
{
    try
    {
        //Получение объекта представления данных таблицы - то что мы видим
        DataView dv = ds.Tables[0].DefaultView;

        if (dv.Count == 0)
        {
            MessageBox.Show("Пустая таблица!!!", "Внимание!!!",
                            MessageBoxButtons.OK,MessageBoxIcon.Information);
            return;
        }

        //Взять коллекцию столбцов
```

```
            DataColumnCollection cols = ds.Tables[0].Columns;
            DataRow zap = ((DataRowView)(bindingSource1.Current)).Row;
            int i;
            string str = "";

            for (i = 0; i < cols.Count; i++)
            {
                //Если поля NULL ToString() возвращает ""
                str += zap[i] + "  ";
            }

            MessageBox.Show(str);

            // Вычислительные операции по первой текущей записи
            int d = 0;
            bool per = !(Convert.IsDBNull(dv[0][0]) || Convert.IsDBNull(dv[0][1]));
            if (per)d = (int)dv[0][0] + (int)dv[0][1];
            str = string.Format("Сумма первых двух полей = {0}", d);
            MessageBox.Show(str);

            // Вычисление среднего арифметического - Вариант 1
            //dataGridView1.DataSource = null;// Отвязка от datagridView –
            //при работе можно не отвязываться

            int sum = 0, kol = 0;
            for (i = 0; i < dv.Count; i++)
                if (!Convert.IsDBNull(dv[i][1]))
                { kol++; sum += (int)dv[i][1]; }

            if (kol>0) str = string.Format("Среднее арифметическое = {0} {1}",
                            1.0*sum /kol,kol);
            else str = "Данные по полю kod не заполнены";
            MessageBox.Show(str,"Вариант 1");

            // ------------------------------------------
            // Вычисление среднего арифметического - Вариант 2
            sum = 0;
            kol = 0;

            foreach (DataRowView row in dv)
                if (!Convert.IsDBNull(row[1]))
                    { kol++; sum += (int)row[1]; }

            if (kol > 0) str = string.Format("Среднее арифметическое = {0} {1}",
                         1.0 * sum / kol, kol);
            else str = "Данные по полю kod не заполнены";

            MessageBox.Show(str,"Вариант 2");
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }

    }
```

Слід зазначити, що для роботи з даними, які бачить користувач, потрібно використовувати клас представлення **DataView**, який входить в об’єкт таблиці **DataTable**. Об’єкт представлення повертається за допомогою

властивості ***DefaultView***. Наявність нульові поля у таблиці можна перевірити, використовуючи метод ***IsDBNull()*** класу Convert.

На рис. 7.12. наведемо результат виконання програми ADO_MDB.



ID	KOD	NAME	VREM	LOG
0	1	aaaa	01.01.2000	
1	2	aaaa	01.01.2010	
2	2	aaaa	01.01.2011	
3	4	aaaa	01.01.2000	
4	5	aaaa	01.01.2000	
5	6	aaaa	01.01.2000	
6	7	aaaa	01.01.2000	
7	8	aaaa	01.01.2000	
8	9	aaaa	01.01.2000	
9	10	aaaa	01.01.2000	

Рис. 7.12. Результат роботи програми ADO_MDB

7.17. Особливості роботи з базою даних MySQL

У прикладі ADO_MDB продемонстровано основні можливості технології ADO. Приклад ADO_MySQL показує можливості технології ADO .NET для БД MySQL. Оскільки приклад ADO_MySQL містить такий самий набір функцій, як і ADO_MDB, зупинимося на відмінностях та особливостях роботи саме з БД MySQL.

На рис. 7.13. наведемо конструктор форм Designer, за допомогою якого спроектуємо форму.

На відміну від прикладу ADO_MDB, тут додано список listBox3 для виведення баз даних MySQL.

Слід зазначити, що MySQL працює з драйвером ODBC, тому що для немає відповідного провайдера OLE DB. З цією метою в програмному коді необхідно підключити простір імен System.Data.Odbc і відповідно всі класи починаються з префіксу «Odbc».

Наведемо код користувальницької форми Form1, у якому здійснюється з’єднання з БД MySQL. У якості користувача вибрано "root", як сервер "localhost".

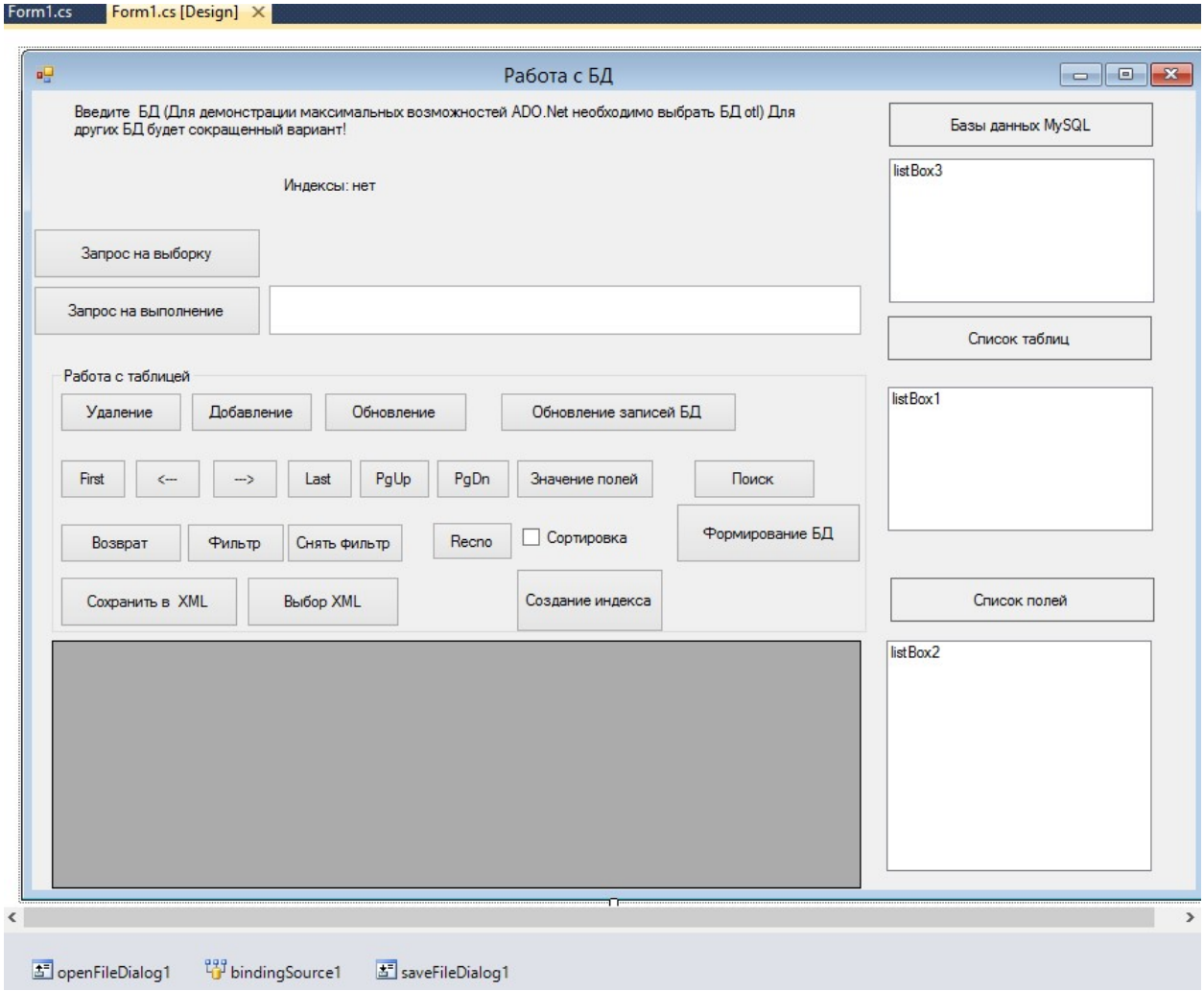


Рис. 7.13. Конструювання додатку ADO_MySQL

```
public Form1()
{
    InitializeComponent();

    conn = null;
    da = null;
    cmd = null;
    ds = null;

   //Формирование объекта Connection
   try
   {
     string source = "DRIVER=MySQL ODBC 3.51 Driver;UID = root;SERVER=localhost";
     conn = new OdbcConnection(source);

    MessageBox.Show(source, "Начальная строка подключения", MessageBoxButtons.OK,
                    MessageBoxIcon.Information);

        conn.Open();

        show(true, 1);
```

```
            if (ds != null) ds.Dispose();
            ds = new DataSet();

            Struct_MySQL();

            dataGridView1.DataSource = null;

            label5.Text = "Индексы : нет";
            kol_index = 0;
        }

        catch (Exception ex)
        {
            if (conn != null && conn.State == ConnectionState.Open) conn.Close();
            if (conn != null) conn.Dispose();
            conn = null;

            if (ds != null) ds.Dispose();
            ds = null;
            show(false);

            MessageBox.Show(ex.Message);
        }
    }
```

На відміну від прикладу роботи з БД MS Access, де в якості бази був окремий файл *.mdb, MySQL у ролі з'єднання (об'єкт Connection) виступає не окремо взята БД, а сервер, до якого підключається певний користувач зі своїми правами. У прикладі до БД підключається головний користувач-адміністратор «root» з максимальними правами доступу. При підключенні використовується раніше встановлений драйвер ODBC «MySQL ODBC 3.51 Driver». Після відкриття з'єднання за допомогою методу Open() об'єкта Connection здійснюється виведення всіх баз даних, що знаходяться на сервері до списку «Бази даних MySQL».

За це виведення відповідає функція Struct_MySQL(), текст якої наведено нижче.

```
// Вывод  БД по MySQL (список таблиц БД MySQL)
void Struct_MySQL()
{
    string str = "show databases";
    //Создание объекта Recordset для чтения
    OdbcCommand cmd = new OdbcCommand(str, conn);
    OdbcDataReader reader = cmd.ExecuteReader();

    listBox3.Items.Clear();

    while (reader.Read())
    {
        str = reader[0].ToString();
        if (str != "mysql" && str != "information_schema")
            listBox3.Items.Add(str);
    }
    reader.Close();
}
```

Для того щоб отримати список доступних БД MySQL необхідно виконати запит СУБД MySQL - show databases (показати бази даних). Такий

запит доцільно виконати за допомогою об’єкта OdbcCommand. При цьому отримаємо об’єкт для читання OdbcDataReader, після чого заповнимо listBox3 базами даних, крім двох системних БД: mysql і information_schema.

Відкриття БД здійснюється при її виборі зі списку (функція відгуку listBox3_SelectedIndexChanged). Наведемо текст цієї функції.

```
// Открытие базы данных(USE) - MySQL
private void listBox3_SelectedIndexChanged(object sender, EventArgs e)
{
    listBox1.Items.Clear();

    baza = listBox3.SelectedItem.ToString().ToUpper();

    OdbcCommand cmd = null;

    try
    {
        //Работа с транзакциями через объект OdbcCommand
        cmd = new OdbcCommand("", conn);
        cmd.Transaction = conn.BeginTransaction();

        // Два запроса для работы с русской кодировкой cp1251 в БД MYSQL
        cmd.CommandText = "SET CHARACTER SET 'cp1251'";
        cmd.ExecuteNonQuery();
        cmd.CommandText = "SET NAMES 'cp1251'";
        cmd.ExecuteNonQuery();
        ////////////////////////////////////////////////////////////
        cmd.CommandText = "USE `" + baza + "`";

        //Выполнить без получения запроса
        cmd.ExecuteNonQuery();
        cmd.Transaction.Commit();
    }

    catch (Exception ex)
    {
        //Откат транзакции
        cmd.Transaction.Rollback();
        MessageBox.Show(ex.Message);
        return;
    }

    Structura_BD();
    textBox1.Text = "";

    kol_index = 0;
    label5.Text = "Индексы : нет";
}
```

Слід зазначити, що для коректної роботи з кирилічним кодуванням у MySQL, необхідно виконати наступні два SQL-запити.

```
.....................................................
            "SET CHARACTER SET 'cp1251'"
            "SET NAMES 'cp1251'"
.....................................................
```

Далі БД відкривається за допомогою команди USE `Ім’я БД`. У синтаксисі MySQL під час роботи з базами даних та таблицями, їх необхідно

брати в одинарні лапки "`", на відміну від БД MS ACCESS, де таблиці беруться у квадратні дужки "[" та "]".

На рис. 7.14. наведемо результат виконання програми ADO_MySQL.

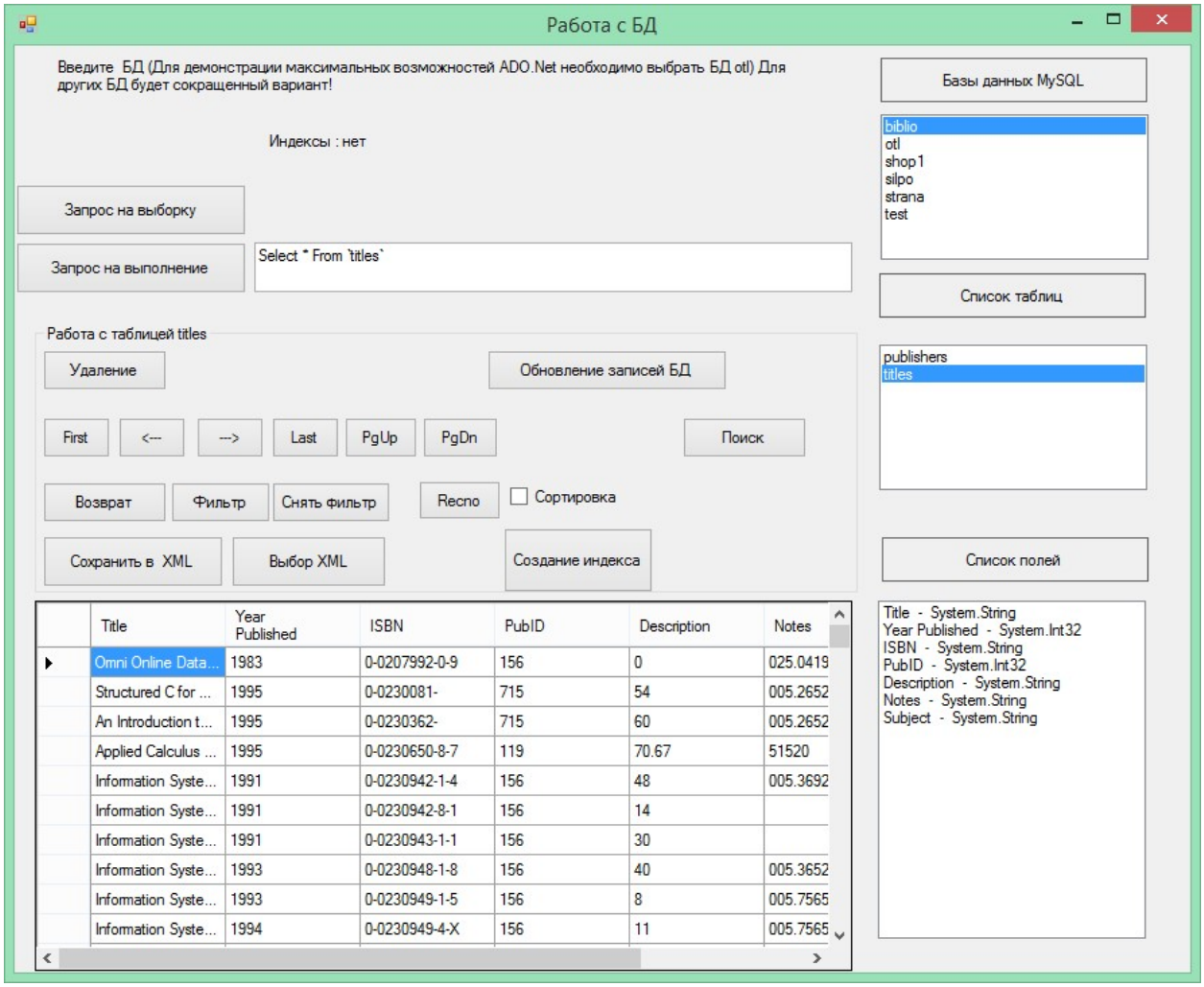


Рис. 7.14. Результат роботи програми ADO_MySQL

7.18. Особливості роботи з базою даних SQL Server

Наведемо особливості роботи з БД SQL_Server з урахуванням прикладу ADO_SQL_Server. Як і в прикладі ADO_MySQL, у прикладі є список БД SQL_Server, які наведено у списку listBox3. Для СУБД SQL Server, фірмою Microsoft розроблено спеціальний .NET-провайдер, адаптований для її оптимальної роботи. Для роботи з SQL Server необхідно підключити простір імен System.Data.SqlClient і відповідно всі класи починаються з префіксу Sql.

Наведемо код конструктора форми Form1, де здійснюється з'єднання з БД SQL Server. У якості СУБД SQL Server обрано версію SQL Server Express 2008, яка входить у постачання Microsoft Visual Studio 2010.

```
public Form1()
{
    InitializeComponent();

    conn = null;
    da = null;
    cmd = null;
    ds = null;

    //Формирование объекта Connection
    try
    {
        string source = @"Data Source=.\SQLEXPRESS;Integrated Security=True";

       //string source = @"Data Source=.\SQLEXPRESS;Integrated
       //Security=True;AttachDbFilename=D:\Example
       //VS.NET\ADO.Net\New\Ado_SQL_Server\Ado3\bin\x86\Debug\Database1.mdf";

     conn = new SqlConnection(source);

    MessageBox.Show(source, "Начальная строка подключения", MessageBoxButtons.OK,
                          MessageBoxIcon.Information);

        conn.Open();

        show(true, 1);

        if (ds != null) ds.Dispose();
        ds = new DataSet();

        Struct_SQLServer();

        dataGridView1.DataSource = null;

        label5.Text = "Индексы : нет";
        kol_index = 0;
    }

    catch (Exception ex)
    {
        if (conn != null && conn.State == ConnectionState.Open) conn.Close();
        if (conn != null) conn.Dispose();
        conn = null;

        if (ds != null) ds.Dispose();
        ds = null;
        show(false);

        MessageBox.Show(ex.Message);
    }
}
```

У рядку підключення Connection використовуємо два параметри: DataSource - підключення до сервера БД (для використання SQL SERVER Express використовується ". \ SQLEXPRESS"); Integrated Security – автентифікація користувача (true – через операційну систему Windows, false – через СУБД SQL Server).

Після відкриття з’єднання за допомогою методу Open() об’єкта Connection здійснюється виведення всіх баз даних, що знаходяться на сервері до списку «Бази даних SQL_Server».

За це виведення відповідає функція Struct_SQLServer(), текст якої наведено нижче.

```
// Вывод  БД по SQL Server (список таблиц БД SQL Server)
void Struct_SQLServer()
{

    DataTable table = conn.GetSchema("Databases");
    //Формирование списка баз данных SQL SERVER

    listBox3.Items.Clear();

    foreach (DataRow row in table.Rows)
        listBox3.Items.Add(row[0]);

}
```

За допомогою методу GetSchema() об’єкта Connection отримаємо інформацію про всі бази даних SQL Server. У якості входу використовується рядковий параметр «Databases». Цей параметр використовується під час роботи з драйверами OLE DB та SQL Server. При роботі з драйверами ODBC цей параметр відсутній, тому при роботі з MySQL ми скористалися запитом show databases замість методу GetSchema().

Відкриття БД здійснюється при її виборі у списку (функція відгуку listBox3_SelectedIndexChanged). Наведемо текст цієї функції.

```
// Открытие базы данных(USE) - SQL_Server
private void listBox3_SelectedIndexChanged(object sender, EventArgs e)
{
    listBox1.Items.Clear();

    baza = listBox3.SelectedItem.ToString().ToUpper();
    string str = "USE [" + baza + "]";
    try
    {
        //Работа с транзакциями через объект SqlCommand
        if (cmd != null) cmd.Dispose();
        cmd = new SqlCommand(str, conn);
        cmd.Transaction = conn.BeginTransaction();
        //Выполнить без получения запроса
        cmd.ExecuteNonQuery();
        cmd.Transaction.Commit();
    }

    catch (Exception ex)
    {
        //Откат транзакции
        cmd.Transaction.Rollback();
        MessageBox.Show(ex.Message);
        return;
    }

    Structura_BD();
    textBox1.Text = "";

    kol_index = 0;
    label5.Text = "Индексы : нет";
}
```

Далі БД відкривається за допомогою USE [Ім’я БД]. У синтаксисі SQL Server під час роботи з базами даних та таблицями, їх потрібно брати в квадратні дужки «[» і «]», як і в СУБД MS ACCESS.

На рис. 7.15. наведемо результат виконання програми ADO_SQL_Server.

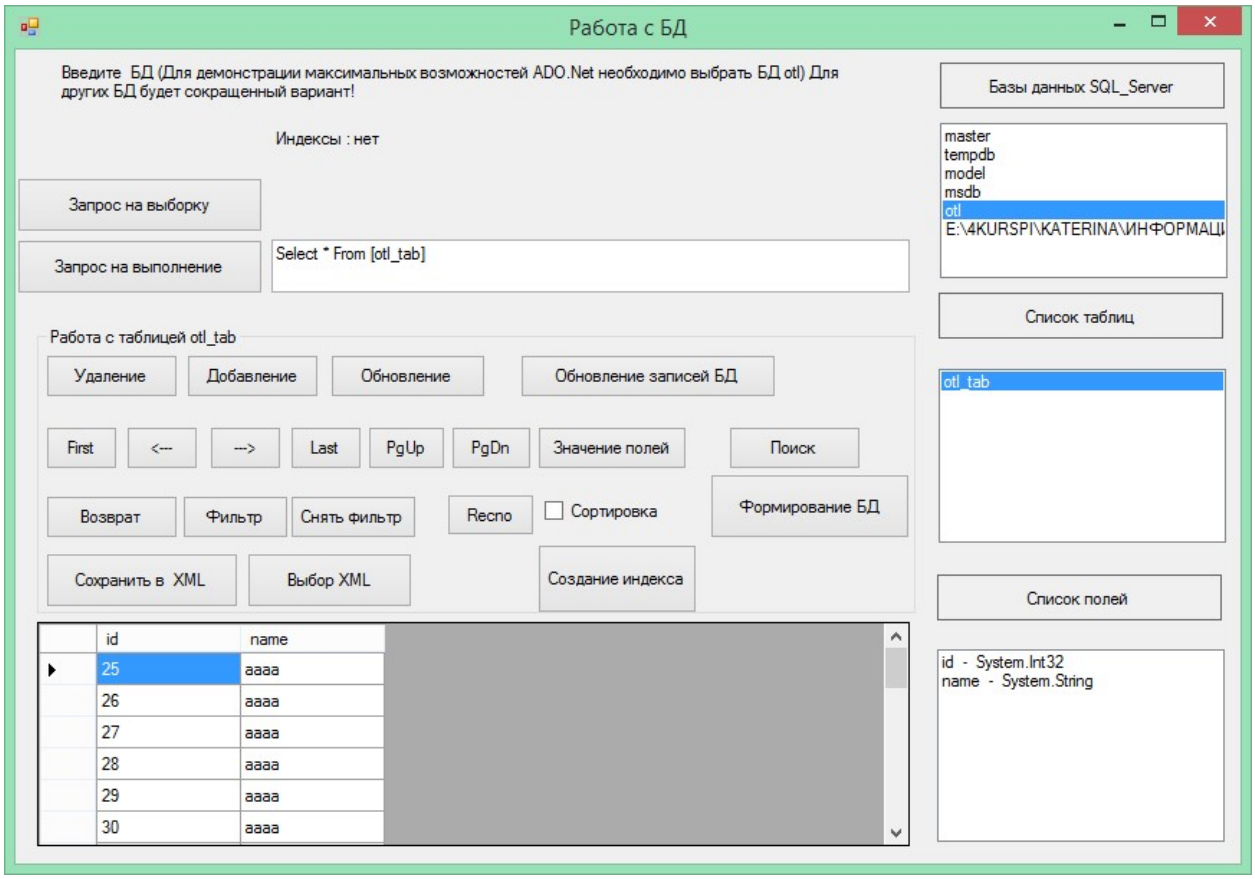


Рис. 7.15. Результат роботи програми ADO_SQL_Server

У прикладах, розглянутих вище, показано основні можливості роботи з технологією ADO .NET. Далі розглянемо додаткові приклади, які розкривають способи прив’язки до картинок (елемент PictureBox), елементів роботи з датою і часом DateTimePicker; способи прив’язки до контейнерів даних, а також можливості застосування різних типів полів елемента DataGridView.

7.19. Приклад ADO_PictureBox

Для цього прикладу використовується таблиця стандартної БД «Борей.mdb», яка йде у постачанні з MS ACCESS. У цьому прикладі відбувається перегляд таблиці «Типи», в якій міститься картинка. Зображення записується у тип поля «Поле об’єкта OLE». При вставці нового об’єкта у полі таблиці БД вибираємо тип об’єкта Bitmap Image. На рис. 7.16 наведемо конструктор таблиці "Типи" БД "Борей.mdb".

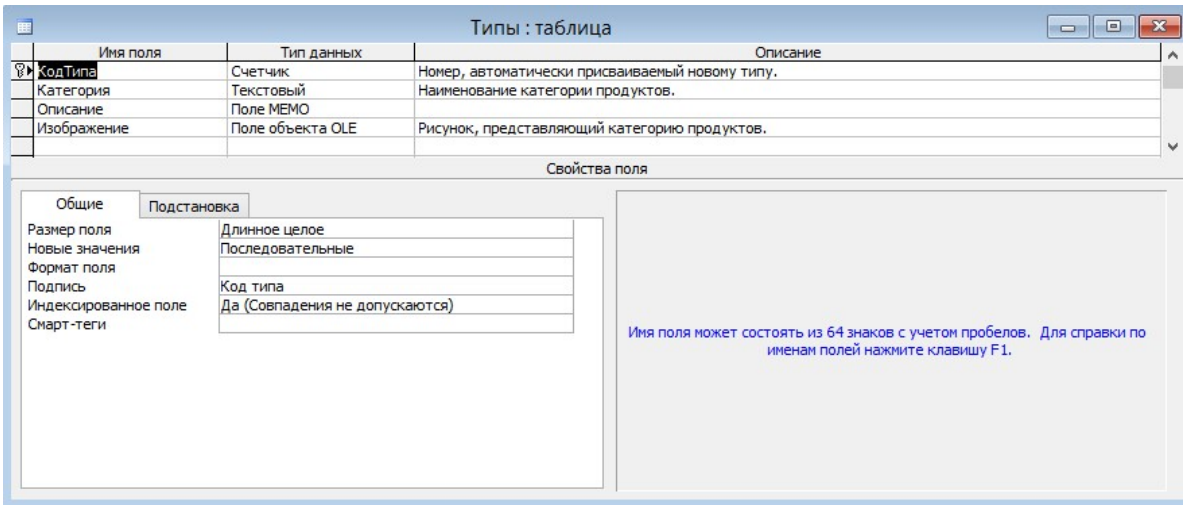


Рис. 7.16. Конструктор таблиці «Типи»

Наведемо програмний код цього прикладу.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OleDb;

using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Ex1
{

    public partial class Form1 : Form
    {

        //Объект Connection с драйвером OleDb
        OleDbConnection conn;
        OleDbDataAdapter da;
        DataSet ds;

        public Form1()
        {
            InitializeComponent();

            //Инициализация и открытие БД, а также привязка таблицы к DatagridView
            //Формирование объекта Connection

            try
            {
                string source = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Борей.mdb";
                if (conn != null) conn.Dispose();
                conn = new OleDbConnection(source);
                conn.Open();
```

```
                da = new OleDbDataAdapter("Select * From Типы", conn);
                ds = new DataSet();

                da.Fill(ds);

                bindingSource1.DataSource = ds.Tables[0];
                dataGridView1.DataSource = bindingSource1;
                bindingNavigator1.BindingSource = bindingSource1;

               //Привязка picturebox к изображению
               pictureBox1.DataBindings.Add("Image", bindingSource1, "Изображение",true);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

У цьому фрагменті спочатку відбувається з'єднання з БД «Борей.mdb», після чого за допомогою методу Fill() об'єкта DataAdapter заповнюється таблиця контейнера DataSet результатом SQL-запиту. Потім відбувається прив'язка даних об'єкта DataTable до об'єктів bindingNavigator1 та dataGridView1 через універсальний об'єкт-зв'язувач BindinSource - bindingSource1.

На наступному етапі прив'яжемо об'єкт bindingSource1 до об'єкта-зображення pictureBox1 за допомогою властивості DataBindings.

```
pictureBox1.DataBindings.Add("Image", bindingSource1, "Изображение",true);
```

Тут "Image" - це властивість для зв'язування з картинкою, "bindinSource1" – зв'язок з об'єктом DataTable, "Зображення" - поле таблиці, true - параметр форматування, який відкидає початкові дані шапки поля "Поле об'єкта OLE" і безпосередньо бере картинку.

На рис. 7.17. наведемо результат виконання програми ADO_PictureBox.

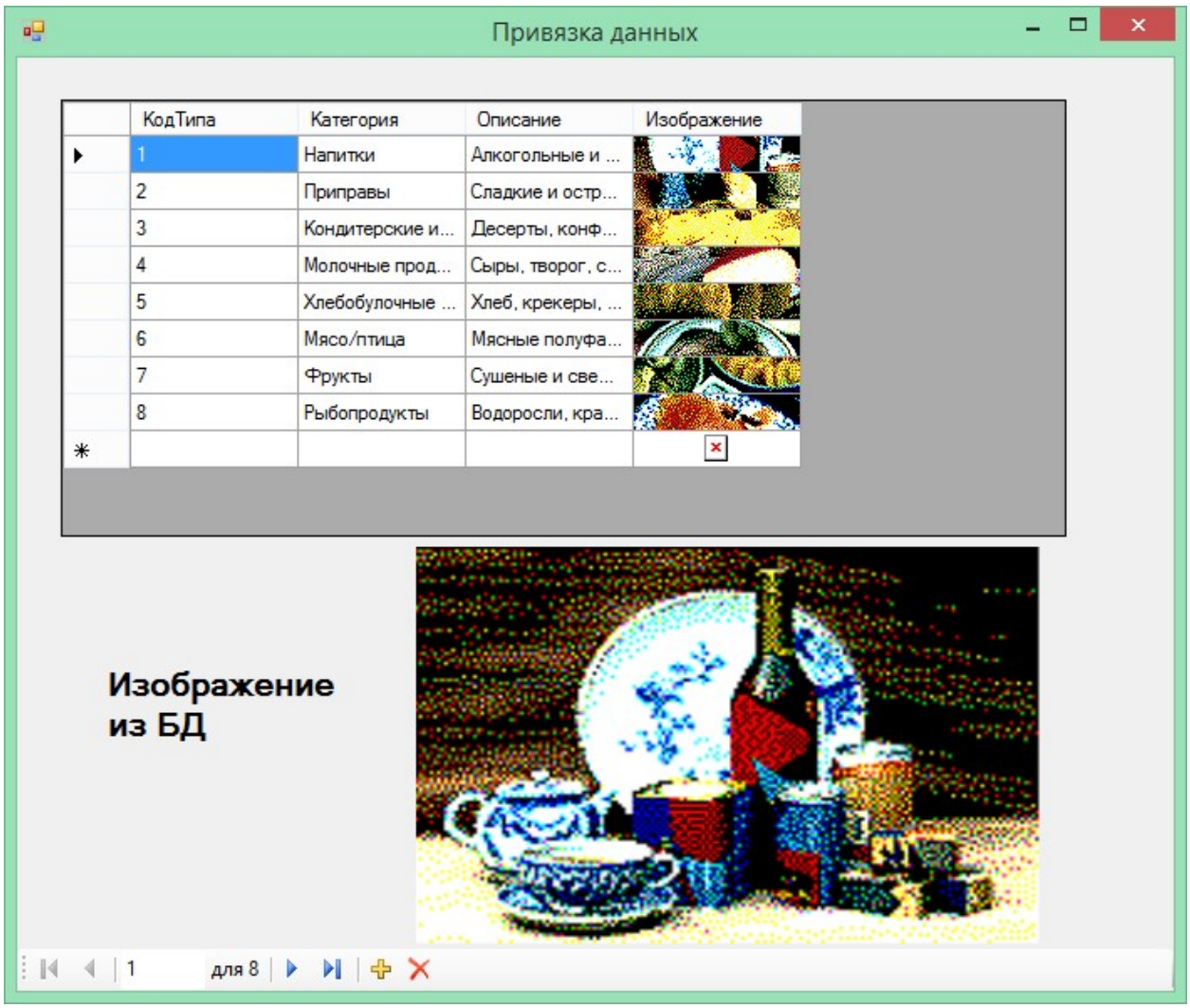


Рис. 7.17. Результат роботи програми ADO_PictureBox

З рисунка видно, що зображення відображається як в елементі DataGridView (тип поля DataGridViewImageColumn), так і в окремому елементі PictureBox, прив’язаному до поля «Зображення».

7.20. Приклад ADO_DateTimePicker

Для цього прикладу використовується таблиця стандартної БД «Борей.mdb», яка йде у постачанні з MS ACCESS. Також у цьому прикладі відбувається перегляд таблиці «Співробітники», в якій міститься поле дати та часу (Дата народження співробітника). На рис. 7.18 наведено конструктор таблиці "Співробітники" БД "Борей.mdb".

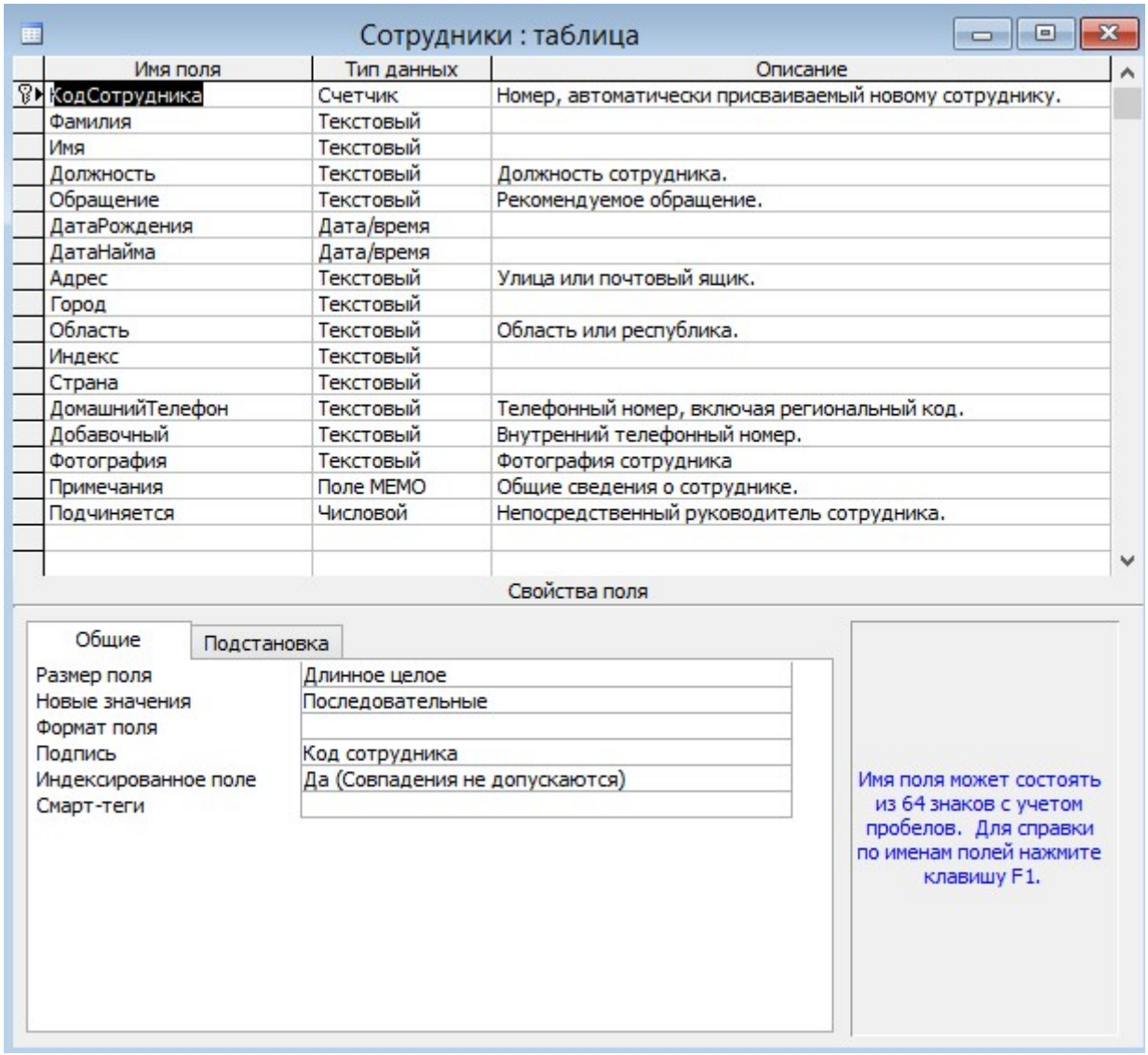


Рис. 7.18. Конструктор таблиці «Співробітники»

Наведемо програмний код цього прикладу.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OleDb;

using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Ex1
{
    public partial class Form1 : Form
    {

        //Объект Connection с драйвером OleDb
        OleDbConnection conn;
        OleDbDataAdapter da;
        DataSet ds;
```

```
        public Form1()
        {
            InitializeComponent();
            //Инициализация и открытие БД, а также привязка таблицы к DatagridView
            //Формирование объекта Connection
            try
            {
                string source = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Борей.mdb";
                if (conn != null) conn.Dispose();
                conn = new OleDbConnection(source);
                conn.Open();

                da = new OleDbDataAdapter("Select * From Сотрудники", conn);

                ds = new DataSet();
                da.Fill(ds);

                bindingSource1.DataSource = ds.Tables[0];
                dataGridView1.DataSource = bindingSource1;
                bindingNavigator1.BindingSource = bindingSource1;

         //Привязка datetimePicker к дате рождения
         dateTimePicker1.DataBindings.Add("Value", bindingSource1, "ДатаРождения");
            }

            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

У цьому фрагменті спочатку відбувається з’єднання з БД «Борей.mdb», після чого за допомогою методу Fill() об’єкта DataAdapter заповнюється таблиця контейнера DataSet результатом SQL-запиту. Потім відбувається прив’язка даних об’єкта DataTable до об’єктів bindingNavigator1 та dataGridView1 через універсальний об’єкт-зв’язувач BindinSource - bindingSource1.

На наступному етапі прив’яжемо об’єкт bindingSource1 до об’єкта-зображення dateTimePicker1 за допомогою властивості DataBindings.

```
dateTimePicker1.DataBindings.Add("Value", bindingSource1, "ДатаНародження");
```

Тут "Value" - властивість для зв’язування зі значенням дати і часу, "bindinSource1" – зв’язок з об’єктом DataTable, "ДатаНародження" - поле таблиці.

На рис. 7.19. наведемо результат виконання програми ADO_DateTimePicker1.

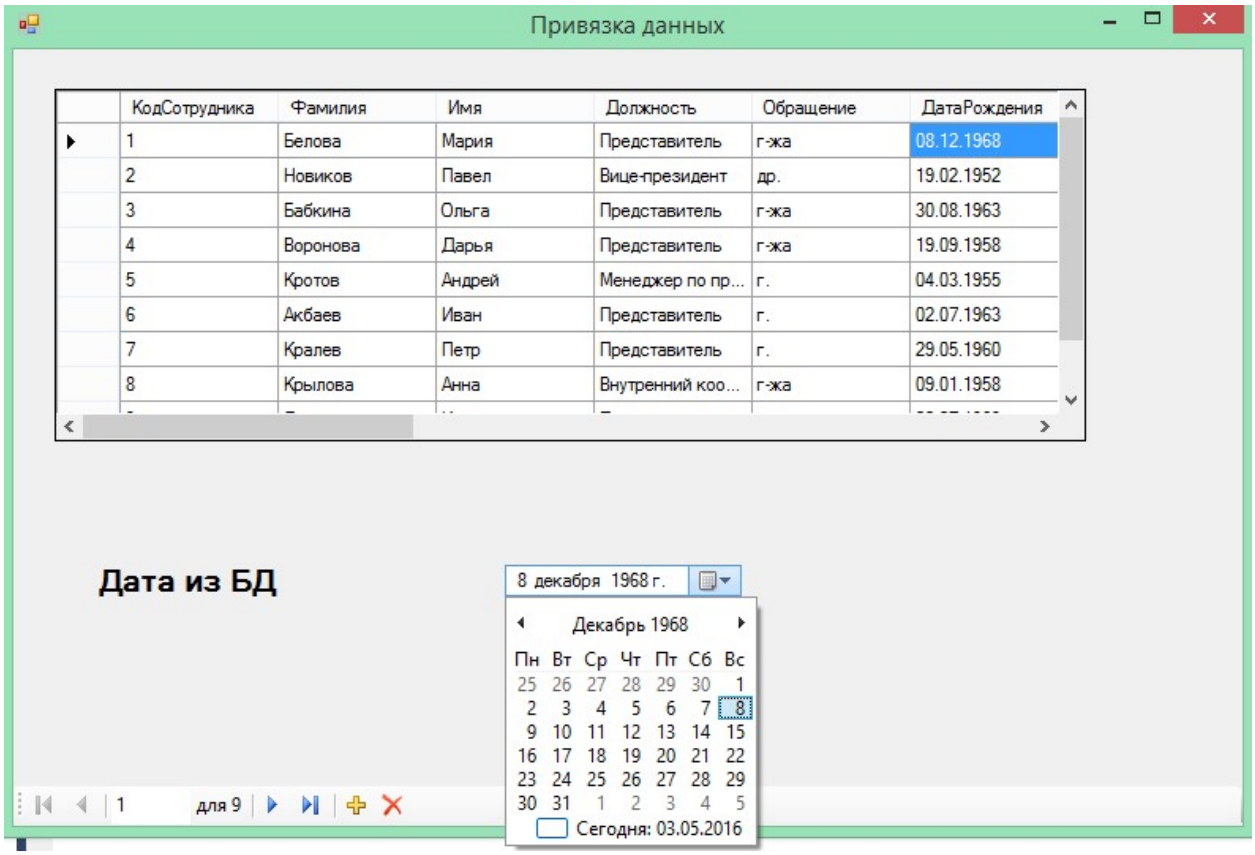


Рис. 7.19. Результат роботи програми ADO_DateTimePicker1

З рисунка видно, що поле "ДатаНародження" відображається як в елементі DataGridView (тип поля DataGridViewTextBoxColumn), так і в окремому елементі DateTimePicker, прив'язаному до поля "ДатаНародження".

Слід зазначити, що елемент DataGridView не має вбудованого елемента DateTimePicker, тому відображення відбувається у звичайному текстовому полі (тип поля DataGridViewTextBoxColumn). Тому для відображення елемента DateTimePicker доцільно застосовувати цей підхід.

7.21. Приклад ADO_LIST

При використанні універсальних елементів прив'язки технології ADO .NET, таких як BindingSource, BindingNavigator, DataGridView – ми працювали з об'єктом DataTable, отриманого в результаті SQL-запиту на вибірку. Але ці елементи можуть працювати не тільки з об'єктом DataTable, але і з контейнерами даних. Одним з ефективних типізованих контейнерів є контейнер List<> (аналогія цього контейнера – vector<> C++).

На початковому етапі створимо клас Info з єдиним цілим полем:

```
class Info
    {
        private int i;

        public Info() {}
```

```
        public Info(int i)
        { this.i = i; }

        public int Number
        {
            get { return i; }
            set { i=value; }
        }

    }
```

Для подальшої прив'язки до елемента BindingSource і роботи з даними необхідно створити властивість, яка буде повертати і приймати дані. Назвемо цю властивість Number. Крім того, для коректного додавання даних необхідно, крім конструктора з параметрами, створити порожній конструктор.

Наведемо клас Form1, у якому здійснюється накопичення даних контейнера, і навіть прив'язка до елементів керування.

```
    public partial class Form1 : Form
    {
        List<Info> l;

        public Form1()
        {
            InitializeComponent();
            try
            {
                l = new List<Info>();
                l.Add(new Info(1));
                l.Add(new Info(2));
                l.Add(new Info(3));
                l.Add(new Info(4));

                //Привязка GridView к контейнеру данных List
                bindingSource1.DataSource = l;
                bindingNavigator1.BindingSource = bindingSource1;
                dataGridView1.DataSource = bindingSource1;
            }

            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

          private void button1_Click(object sender, EventArgs e)
          {

              foreach (var el in l)
                  MessageBox.Show(el.Number.ToString());
          }
    }
```

У прикладі оголосимо об'єкт l контейнера List<Info> і заповнимо його даними чотирьох чисел. Потім за допомогою універсального зв'язувача bindingSource1 прив'яжемо цей контейнер до об'єктів bindingNavigator1 і dataGridView1.

Функція відгуку button1_Click відображає елементи контейнера на

екрані. Так, при зміні, додаванні або видаленні даних з DataGridView, відповідні зміни відобразяться в контейнері даних l.

На рис. 7.20. наведено результат виконання програми ADO_LIST.

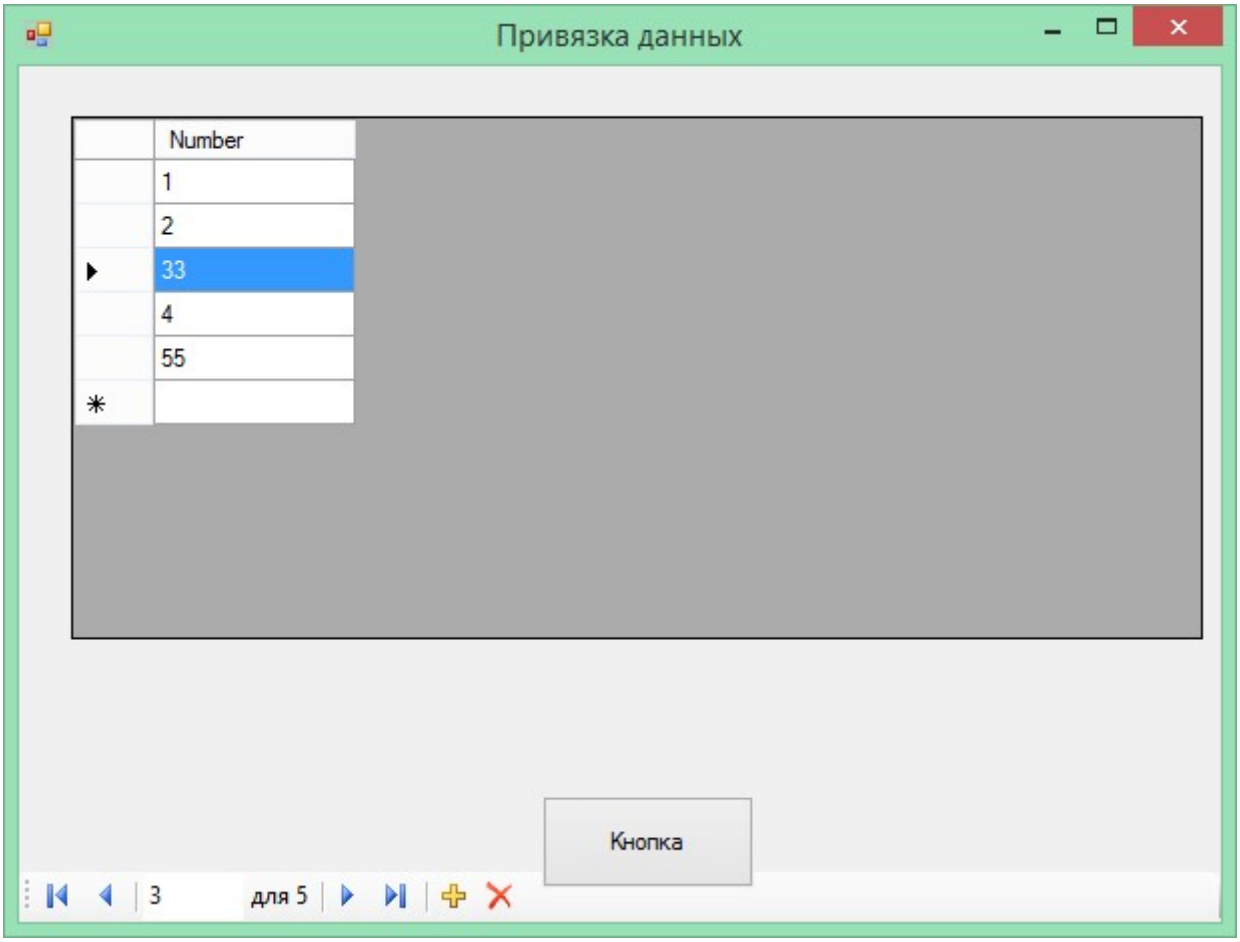


Рис. 7.20. Результат роботи програми ADO_LIST

7.22. Приклад ADO_DataGridView

Метою цього прикладу є демонстрація різних типів полів елемента керування DataGridView. У прикладі ADO_PictureBox вже було продемонстровано поле DataGridViewImageColumn для роботи з зображеннями. Усього існує 6 типів полів:

1. DataGridTextBoxColumn – тип колонки текстового поля. Встановлено за замовчуванням.
2. DataGridViewComboBoxColumn – поле зі списком, що випадає.
3. DataGridViewButtonColumn – поле з кнопкою.
4. DataGridViewLinkColumn – поле з гіперпосиланням.
5. DataGridViewCheckBoxColumn – поле з прапорцем (застосовується для логічних типів полів).
6. DataGridViewImageColumn - поле з картинкою.

У цьому прикладі наведемо код програми, де покажемо 5 типів полів.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OleDb;

using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Ex1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            try
            {
                //Работа с разнотипными столбцами в DatagridView
                //Колонка 1
                //DataGridViewTextBoxColumn;

                dataGridView1.Columns.Add("Колонка1", "Колонка1");
                dataGridView1.RowCount = 5;

                //Колонка 2 - поле со списком
                DataGridViewComboBoxColumn combo = new DataGridViewComboBoxColumn();
                String []mas = { "1", "2", "3", "4", "5" };
                combo.DataSource = mas ;
                combo.Name = "Колонка2";
                dataGridView1.Columns.Add(combo);

                //Колонка 3 - кнопка
                DataGridViewButtonColumn but = new DataGridViewButtonColumn();
                but.Name = "Колонка3";
                dataGridView1.Columns.Add(but);
                //Колонка 4 - гиперссылка
                DataGridViewLinkColumn link = new DataGridViewLinkColumn();
                link.Name = "Колонка4";
                dataGridView1.Columns.Add(link);
                //Колонка 5 - checkbox
                DataGridViewCheckBoxColumn check = new DataGridViewCheckBoxColumn();
                check.Name = "Колонка5";
                dataGridView1.Columns.Add(check);
                //DataGridViewImageColumn - работа с изображением (позже - массив Image)
                //Установка в ячейках начальных значений
                for (int i = 0; i < 5; i++)
                {
                    dataGridView1[0, i].Value = "Привет"+i;
                    dataGridView1[1, i].Value = mas[i];
                    dataGridView1[2, i].Value = "Кнопка"+i;
                    dataGridView1[3, i].Value = "Гиперссылка" + i;
                    dataGridView1[4, i].Value = (i&1)==0;
                }
            }

            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

На рис. 7.21. наведемо результат виконання програми ADO_DataGridView.

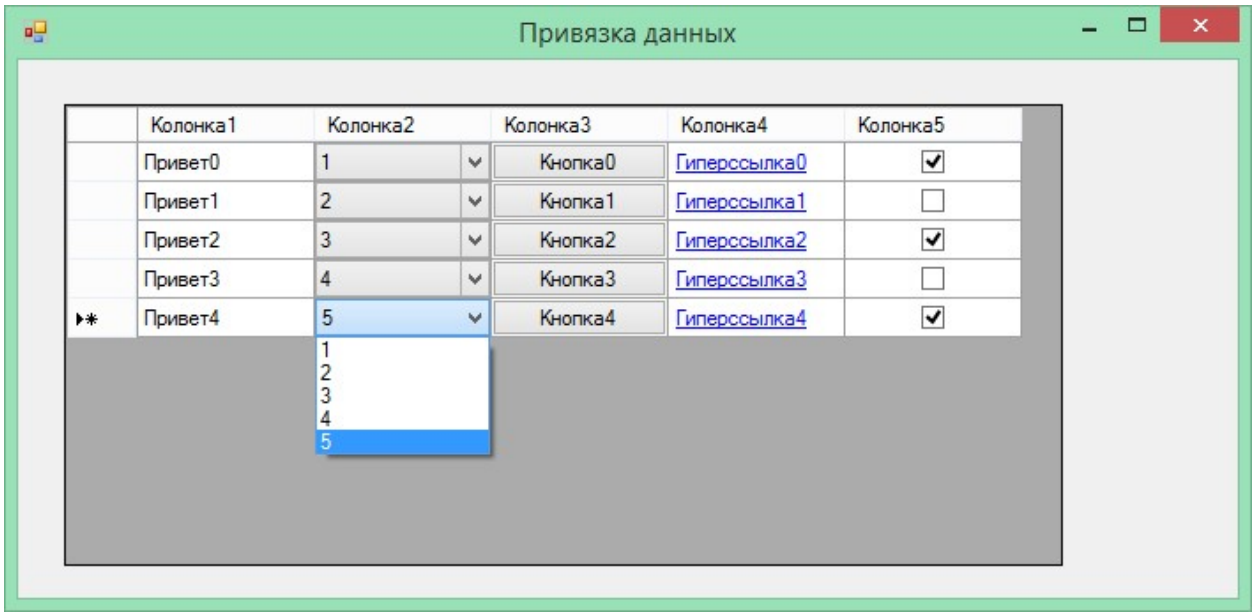


Рис. 7.21. Результат роботи програми ADO_DataGridView

СПИСОК ЛІТЕРАТУРИ

1. Зеленський О.С., Лисенко В.С. Розробка програмного забезпечення на мові C#. Частина 1. Навчальний посібник. - Кривий Ріг: Криворізький економічний інститут Державного вищого навчального закладу "Криворізький національний університет", 2012.- 134 с.

2. Зеленский А.С., Лысенко В.С. Разработка програмного обеспечения на языке C#. Часть 2. Учебное пособие. - Кривой Рог: Криворожский экономический институт ГВУЗ "КНЭУ шимени Вадима Гетьмана", 2015.- 160 с.

3. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 1./Пер. с англ. - М.: Издательско-торговый дом «Русская Редакция», 2002. – 576 с.